ERA-671

NRCC No. 24488

National Research Council Canada
Division of Electrical Engineering

Conseil national de recherches Canada
Division de génie électrique

A VISION-BASED ROBOT SYSTEM USING A MULTIPROCESSING CONTROLLER

S. ELGAZZAR, D. GREEN AND D. O'HARA

85-3214/

Copyright 1986 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source is fully acknowledged.

Droits réservés

A VISION-BASED ROBOT SYSTEM USING A MULTIPROCESSING CONTROLLER

S. Elgazzar, D. Green, and D. O'Hara

ERB-971
NRCC No. 23485

JUNE 1984

TABLE OF CONTENTS

	Page
ABSTRACT/RÉSUMÉ	v
1. INTRODUCTION	1
2. SYSTEM CONFIGURATION	2
2.1 Physical Organization	2
2.2 Hardware Organization	4
2.3 Software Organization	5
2.3.1 Overview	5
2.3.2 Software development mechanics	6
2.3.3 Software packages	6
2.3.3.1 VALPAK and VALLO	6
2.3.3.2 Ehvt_server	8
2.3.3.3 VGLIB	8
3. VISION APPLICATION PROGRAM	10
3.1 Binary Image Acquisition	10
3.2 Image Processing and Feature Extraction	10
3.2.1 Thresholding	10
3.2.2 Edge detection	11
3.2.3 Feature extraction	11
3.3 Part Recognition	11
3.4 Vision Program	13
4. VISION ALGORITHMS	13
4.1 Thresholding	13
4.2 Edge Detection	13
4.2.1 Boundary following algorithm	13
4.2.2 Thinning algorithms	14
4.3 Feature Extraction	14
4.3.1 Area	14
4.3.2 Center of gravity	17
4.3.3 Perimeter	17
4.3.4 The p2a shape factor	17
4.4 Corner Detection	17
4.5 Orientation	18
5. VISION CALIBRATION AND APPLICATION PROGRAM	18
5.1 Camera Calibration	19
5.2 VAL Application Program	20
6. PUTTING IT ALL TOGETHER WITH INTERTASK COMMUNICATIONS	22
7. SUMMARY	24
ACKNOWLEDGEMENTS	24
REFERENCES	25

APPENDICES

		Page
A	Listing of VALPAK, VALLO, Getfloat	27
B	Listing of Ehvt_server	49
C	Listing of VGLIB	55
D	Listing of Vision Application Program	63
E	Listing of Vision Subroutines	73
F	Listing of Robot Application Program	91

LIST OF FIGURES

1	The Vacuum Gripper	3
2	The Robot Work Surface	3
3	Hardware Organization	5
4	The Message-Passing Sequence for VGLIB and its Notifier	9
5	The Decision Tree	12
6	Flowchart for Boundary Following 'bdryb'	15
7	Flowchart for Boundary Following 'bdrya'	16
8	Orientation for Pentagon	19
9	Definition of Camera Frame Peferred to Robot Base Frame	19
10	VAL Sequence for Block Grasping and Insertion	22
11	Summary of Intertask Communication	23

ABSTRACT

This report describes the implementation of a sensor-based robot using a binary vision system. The vision system locates and recognizes regular shaped blocks which are then inserted into matching sockets. The vision algorithms and the control software implementation on a multiprocessor system are described. This system of parallel processors permits the concurrent execution of vision analysis and robot motion control tasks. The procedure for vision calibration and the sequence of VAL commands used to drive the PUMA 560 arm are also described.

RÉSUMÉ

Le présent rapport décrit la mise à l'oeuvre d'un robot à système de vision binaire. Ce système permet au robot de localiser et de reconnaître des blocs de forme régulière qu'il insère ensuite dans des réceptacles de formes correspondantes. Sont aussi décrits les algorithmes de vision et l'implantation du logiciel de commande sur un système multiprocesseur. Ce système, à processeurs parallèles, rend possible l'exécution simultanée de l'analyse visuelle et de la commande des gestes du robot. La méthode d'étalonnage visuel et la séquence des commandes VAL du bras PUMA 560 sont expliquées.

A VISION BASED ROBOT SYSTEM USING A MULTIPROCESSING CONTROLLER

S. Hignasar, D. Green, and D. O'Hara

CHAPTER 1. INTRODUCTION

This report describes the implementation of a robot assembly task using a two-dimensional binary vision system.

The robot application is one in which the robot picks a child's block from an indexed circular tray and inserts it into the socket of matching shape on a "puzzle tray". The selection of this application was made to appeal to the general public, because the set-up was a display in the laboratory "Open House '83". Although the details would change, this application addresses all the fundamental problems of applying binary vision to industrial robotics. The blocks could be small parts which are to be sorted, inspected, and packaged.

The blocks are manually placed in five of the six sectors of a rotating tray. The sixth empty sector is always located under the puzzle tray. The parts have random position and orientation (pose) within each sector, but cannot be placed on their edge as the vision system does not provide depth information. (A program extension to cope with this situation does not present any fundamental challenge.) The blocks present a smooth face ideally suited to gripping by a small suction cup. The camera views the scene from overhead and takes an image of the sector presented to it. As soon as the vision system identifies a block and determines its pose, the robot manipulator begins its sequence of grasping and insertion while the vision system continues the process of locating and identifying the next block. The puzzle tray has five shaped sockets and any block which does not match is thrown into a "reject" box.

This effort is in support of a broader, long term interest in sensor-based robotics. We are investigating the systems aspects of robotics from a computer technology perspective. This includes those structures, both hardware and software, related to the control of complex tasks in loosely constrained environments. The vision system is typical of advanced sensors which require significant computational resources. In this application, the multiprocessor system enables the robot to begin the assembly before the vision system has completed the analysis of the whole scene; parallel processors provide simultaneous operation.

The system configuration is presented in Chapter 2. Chapters 3 and 4 describe the vision application program for shape recognition and the vision algorithms, respectively. The description of the system calibration

and the robot application program can be found in Chapter 5. Listings of the different programs written in conjuction with this project are available in six appendices at the end of the report.

A short video tape (U-matic) of this demonstration is available and can be obtained on loan from the authors.

CHAPTER 2. SYSTEM CONFIGURATION

This chapter outlines the organization of the system. It describes the physical layout of the equipment, the hardware organization, and the software organization as well.

2.1 Physical Organization

Central to the demonstration is a PUMA 560 robot arm manufactured by Unimation. This is an articulated arm with six rotary joints driven by dc servos. The arm is supplied with its own controller, which includes the VAL programming language (1). The controller also provides an I/O module consisting of a number of output relays that can be used for on-off control of motors or other devices and input relays to sense switch closures.

Attached to the arm is a custom designed vacuum gripper which includes a spring-loaded tip (Fig. 1). A microswitch attached to the gripper closes when the tip is pushed beyond a safe limit. This switch is interfaced to the I/O module and brings the arm to an immediate halt if it attempts to drive the gripper into a fixed surface. The vacuum for the gripper is derived from a venturi that is attached to a compressed air supply. An electric solenoid connected to the I/O module controls the air supply and hence the vacuum itself.

The work surface consists of a motor-driven circular table that is divided into six sectors (Fig. 2). The motor is controlled through some electronics such that the I/O module will turn it on but it will stop only when a photodetector recognizes one of six index holes. This ensures that each sector is correctly aligned after the table has been rotated.

A puzzle tray is mounted above the circular table and is aligned with one of the sectors. This tray contains five different shaped holes which correspond to the shapes of the blocks to be acquired. When the robot correctly aligns a block with one of the holes, the vacuum is released and the block will fall through the hole and onto the surface of the circular table.

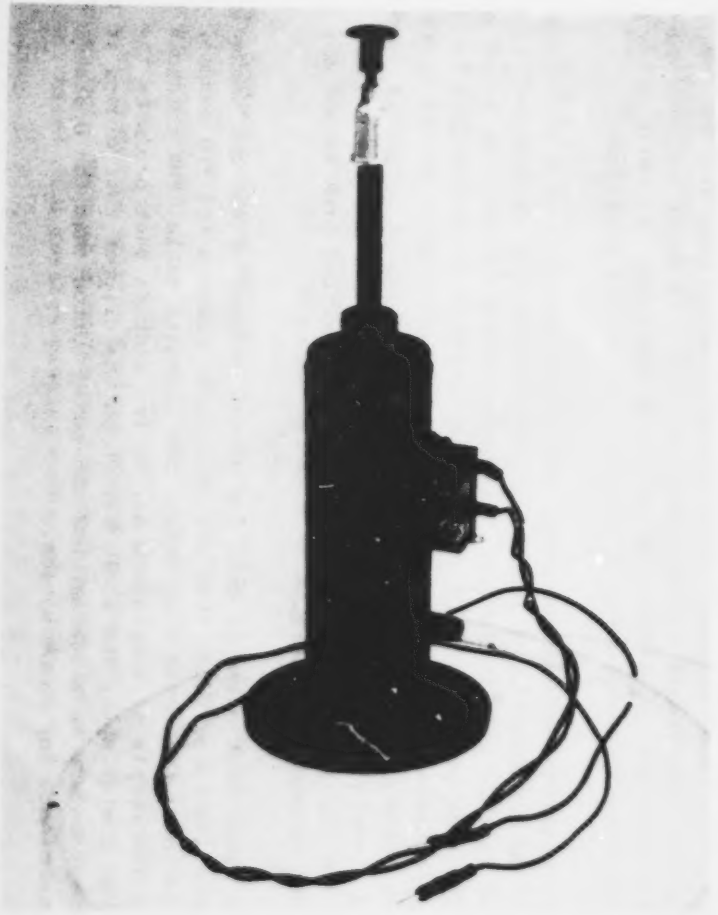

Fig. 1. The vacuum gripper

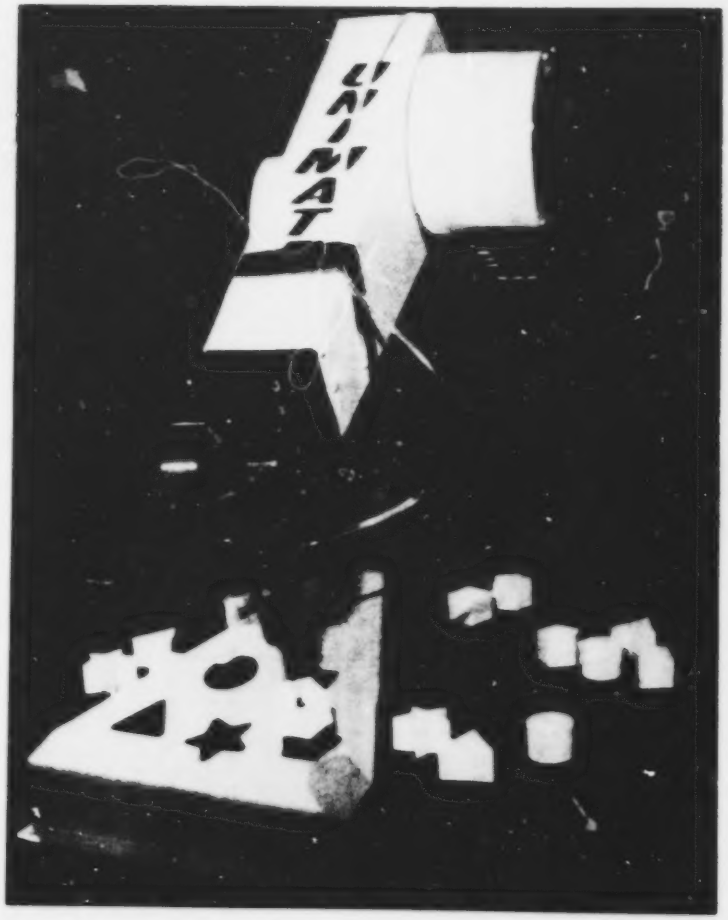

Fig. 2. The robot work surface

A CCD television camera (CCD4000), manufactured by Fairchild and fitted with a zoom lens, is mounted directly above one of the sectors of the circular table. (Vidicon cameras suffer from high distortion and therefore are unable to provide sufficient accuracy over the field of view.) The focal length of the lens is adjusted so that the blocks lying within that sector just fill the field of view of the camera. To minimize parallax errors, the camera is secured to a ceiling-mounted frame placed well above the work surface. This frame also supports a number of spotlights to provide the necessary lighting.

For safety reasons, a "panic button" has also been added that will bring any robot motion to an immediate halt. This switch is connected to the I/O module.

2.2 Hardware Organization

The hardware organization of the system is illustrated in Fig. 3. The system is controlled by the Chorus multiprocessor that was developed in-house (2). Chorus is a multiprocessor architecture that employs 68000-based single board computers (OB68K1A manufactured by Omnibyte). Currently two processors identified as processor 0 and processor 1 are being used. The system has been built on the IEEE-796 Microcomputer System Bus (Multibus). This bus was chosen primarily because of the large number of vendors that supply components for it.

The system also includes a frame grabber (VG-121 manufactured by Datacube (3)). The frame grabber, which is connected to both the CCD camera and a monitor, can freeze a video frame in real-time under computer control. Although the image data are stored in the frame grabber's private memory, they can be accessed from the Multibus by any processor.

The interface between the robot and Chorus consists of a serial RS-232 line that is connected to a serial port on processor 1. A terminal is also connected to this line to monitor the communication between the robot and Chorus for test purposes. Also, with this terminal it is possible to manually intervene in the control of the robot. Two other serial lines are currently being used on processor 0. A user terminal is connected to one port and a serial line to a VAX-11/780 is connected to the other. The VAX is used for all software development and the resulting core images are downloaded through this port.

The robot controller is used for direct control of all the mechanical components including those that are interfaced through the I/O module. The latter include: the motor that drives the circular table, the solenoid that controls the supply of compressed air to the venturi that produces the vacuum for the gripper, the limit switch on the gripper that closes if the gripper encounters an obstacle, and the panic button that the operator can use to terminate any robot motion.

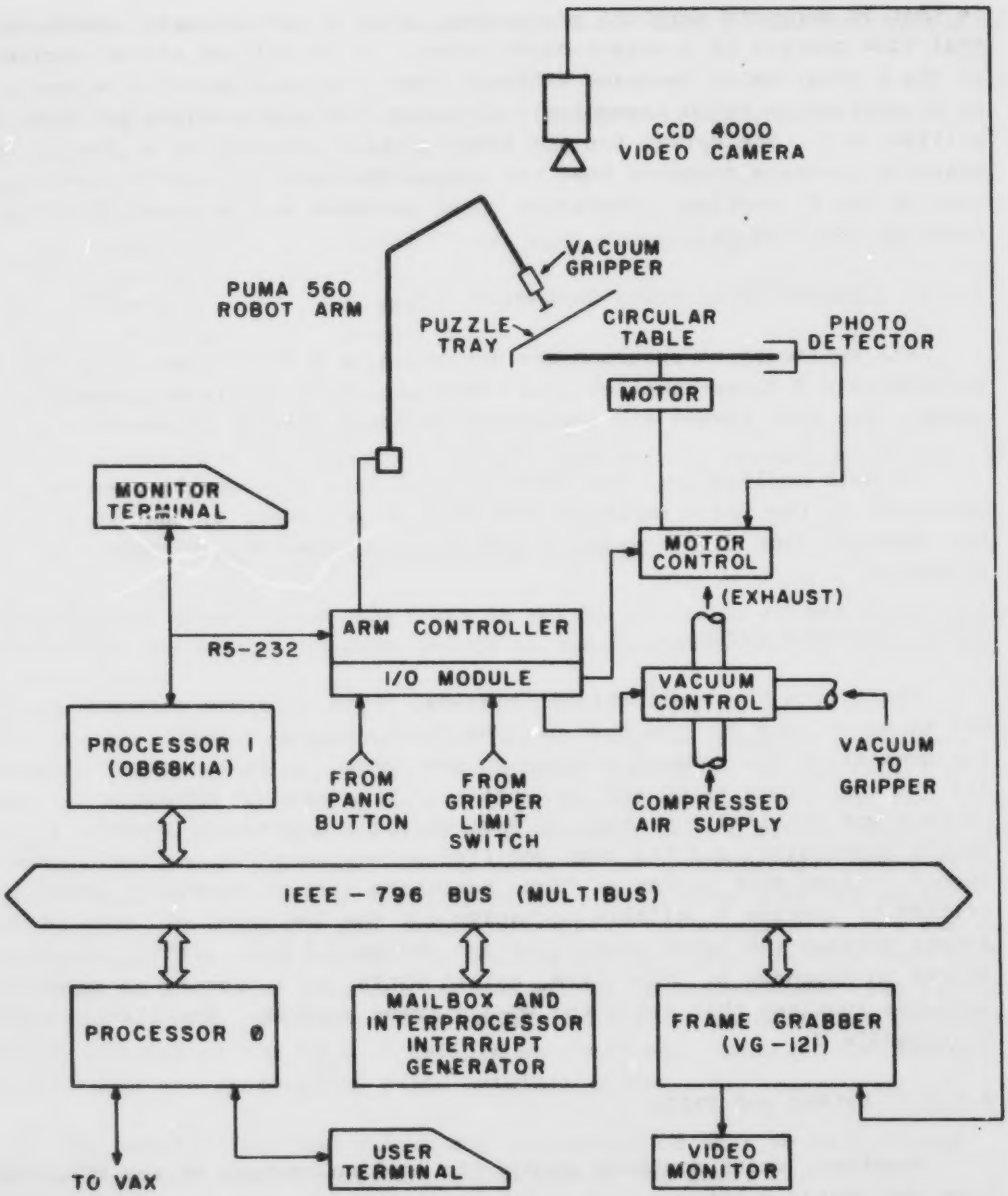

Fig. 3. Hardware organization

2.3 Software Organization

2.3.1 Overview

All software written for the system runs under the Harmony operating system (4). This is a real-time multitasking, multi-processor operating system based on a message-passing mechanism. Harmony's principal advantage

is that it supports parallel processing; this is particularly important for real-time control of a sensor-based robot. It is written almost entirely in the C programming language although some processor specific segments had to be written in 68000 assembler. Likewise, all applications programs are written in C. The driver for the frame grabber consists of a library of assembly language routines that are compatible with the calling conventions used by the C compiler. Therefore these routines are accessed directly as function calls in C.

2.3.2 Software Development Mechanics

All applications programs are developed on a VAX-11/780 using the Whitesmith's C cross-compiler (5). Each processor requires a separate core image. The core images are downloaded directly through processor 0 (2).

In this application, the robot control task ("program") runs on processor 0, the image analysis task runs on processor 1, the I/O task for the terminal runs on processor 0, and the I/O task for the robot runs on processor 1.

2.3.3 Software Packages

Apart from the applications programs, three separate software packages had to be written for the system. The first, called VALPAK, was required for generating the command strings to the robot. A full listing of VALPAK and its companions VALLO and Getfloat() is presented in Appendix A. This package was originally written by Paul Merrill, a graduate student at McGill University, and has been modified extensively for use here. The second, called Ehvt_server or "Echo handshake virtual terminal" server, was written to provide a reliable mechanism for the low level exchange of characters between the robot controller and Chorus. A full listing is presented in Appendix B. The third, called VGLIB, is a library of assembly language routines that drive the VG-121 frame grabber. Its listing appears in Appendix C.

2.3.3.1 VALPAK and VALLO

Together, these routines permit the on-line control of the PUMA robot from the Chorus multiprocessor. VAL permits the user to remotely control the robot from a terminal with the "DO" command (1). By emulating a terminal, VALPAK allows a user program to do the same. VALLO consists of a number of low level routines that handle the character I/O with the robot. It is designed to transmit character strings to the robot controller, to collect the robot's response in a character buffer, to identify the robot response to each command, and to assign an appropriate integer-valued code to each response. This code is returned automatically after each VALPAK function call. VALLO will not only identify error conditions arising from

the robot controller but also extract numerical data transmitted by the robot.

VALPAK implements the following VAL program instructions:

align	moves
appro	set
appros	setc1 ("set" with compound transformation)
delay	shift
depart	signal
departs	speed
drive	where
move	

To illustrate the design of VALPAK consider the speed() function. Its calling syntax is:

```
rply_int = speed(speed_value, message).
```

The relative speed of the robot arm is set to the value of the first argument while the second argument points to the character string that is the robot's reply.

A full listing of the function can be found in Appendix A within VALPAK. First an array, in_str, is stuffed with nulls by the fill() function and then filled with a formatted string by the decode() function (5). (In C, all character strings must be terminated with a null.) This string now satisfies the VAL syntax and is ready to be transmitted. Next, the connections to the robot's serial port are established with the Harmony functions _Selectinput() and _Selectoutput(). The argument to the function send_rb() is a pointer to the VAL message string. The function send_rb() is found in VALLO and ensures that the string is transmitted. The function get_msno(), which is also part of VALLO, returns an integer that indicates either success or the type of failure encountered. Finally, the serial connections are restored to their original state.

The setc1() function allows the assignment of what we call "first level" compound transformations, which are of the form:

```
newtrans = sometrans:relativetrans.
```

The use of "second level" transformations would require the preparation of a setc2() function, etc.

The only other departure worthy of comment is the where() function. This is a command that interrogates the robot to determine its current position. The function returns 12 floating point values by reference

(i.e., pointers). These include the robot's cartesian or "world" coordinates (x y z position and wrist orientation angles) as well as the six joint angles. The function includes a call to the VALLO function get_str() that extracts the numerical data from the robot response. These data are packed into the string array in_str and must now be converted to floating point values. Unfortunately the library support for the cross-compiler included a number of functions that simply did not work, including getfmt(), which is used for formatted input. Consequently the function Getfloat() was written; it converts a numerical string into a floating point number. If Getfloat() detects an error in the string, it sets the corresponding floating point number to 0.0 and issues a warning, "getfloat returns 0".

2.3.3.2 Ehvt_server

The "Echo handshake virtual terminal" server was written by Morven Gentleman to provide a reliable mechanism for character transfer with the robot controller. A user program transfers characters to the server with the Harmony _Put() function. However, after the server transfers a character to the output device (_Tto), it waits for the echo before proceeding. This ensures that each character is transmitted successfully. If an unsolicited character is received by the input device (_Tti) and if it is an "interesting" character (alphanumeric or punctuation), then it will be placed in an input buffer. This buffer is transferred to the user's input buffer when a _Get() function is called. The reader is encouraged to review Appendix B for details of the Ehvt_server.

2.3.3.3 VGLIB

This software package (Appendix C) was prepared for the Datacube VG-121 frame grabber so that it could be accessed by user programs written in C. Under program control it allows the user to clear the display memory, to freeze a frame in real-time, to read a digitized image into system memory, to write an image into the display memory, and to write to individual pixels. Other functions such as synchronization mode control and window position control are also included.

The VG-121 operates in an interrupt-driven mode using Harmony level 4 interrupts. A task that runs under Harmony and that is designed to await an interrupt must run at the same priority as the interrupt itself. Therefore it is desirable to include the _Await_int() function (4) in a very small task called a "notifier" whose role is merely to block and await the interrupt. Only when the interrupt arrives should the notifier task attempt to communicate with its correspondent task.

Two functions in VGLIB, acfield() and wash(), are interrupt-driven; clearly both of these require a notifier task. In the applications program

listing (Appendix D) "Taxi" serves as the notifier. Initially the notifier is "send blocked", i.e., blocked on a _Send() (Fig. 4). Before acfield(), for example, is called, a _Reply() is sent to the notifier thereby advancing it so that it is blocked awaiting an interrupt. At this point acfield() is called and after initiating the command on the VG-121 acfield() becomes "receive blocked". When the VG-121 has completed its operation, it raises an interrupt that is immediately recognized by the notifier that unblocks, promptly sends a message to the calling task, and becomes "send blocked" again. Acfield(), having received a message, unblocks and continues. The entire sequence can now be repeated.

MAIN TASK

```
        -
        -

/*  reply to notifier then call
    acfield.  */

        _Reply( - , taxiid );
        acfield( );

                -
                -

/*  the following assembly language
    code is in acfield.
    It initiates a frame grab on the
    VG-121 and then waits for a
    message from the notifier before
    returning.  */

                -
                -
        move.b  #0x24, 1(a0)
        move.b  #0x20, 1(a0)
                -
                -
        jsr __Receive
                -
                -
        rts
```

NOTIFIER TASK

```
        -
        -

/*  send message to main then
    wait for VG-121 interrupt.
    loop forever  */

for( ;; )
{
        -
        -
        _Send( -,-,- );
        _Await_interrupt( - );
}
```

Fig. 4. The message-passing sequence for VGLIB and its notifier

CHAPTER 3. VISION APPLICATION PROGRAM

Vision is an essential part of flexible automation. The application, as defined in Chapter 1, is to fit regular shaped blocks into corresponding sockets in a puzzle tray. The PUMA arm should be able to locate, grasp, and orient the blocks to successfully accomplish the assembly task.

General purpose binary vision subroutines were written to solve this problem. A detailed description of these routines is presented in this chapter and the next; and the listing of all vision programs can be found in Appendices D and E. It is assumed that the parts are not touching and are lying on either of their faces, not on their sides.

The system, as it stands now, contains a minimum set of subroutines that gives the arm the ability to recognize the parts on the basis of their size and shape regardless of their position and orientation. In addition to recognition it computes the position and orientation of a part so that the arm can acquire it.

If, for any other application, more features are needed for recognizing or orienting a part, the corresponding subroutines can easily be added to the existing library of routines.

3.1 Binary Image Acquisition

The vision system operates exclusively with binary images. This has the advantage of reducing the amount of data that must be handled. It also allows the use of the well-developed methods of image analysis.

To obtain good binary images, contrasting backgrounds and illumination are imperative. In this experiment, yellow blocks are placed on a contrasting black background. Control of illumination is used to enhance (or eliminate) shadows and to reduce the effect of reflections from the surrounding surfaces.

3.2 Image Processing and Feature Extraction

3.2.1 Thresholding

The image we are dealing with is a 320 × 240, 64 grey level image. The first algorithm applied is the thresholding algorithm. It is used to transform the grey level image into a binary image. A fixed threshold is selected to give the best discrimination between the blocks and the background. The convention used is to assign "0" to the points of the background and "1" to the points belonging to the object.

3.2.2 Edge Detection

After thresholding, the image is accessed only once. The image is scanned a line at a time, and once a blob (an object) is found the scanning process is interrupted and the contour following starts.

The detected edge coordinates are stored in two arrays x and y. This information is used to extract the features needed for part recognition and acquisition.

3.2.3 Feature Extraction

The features now available are

(i) Position features
x_g, y_g : center of gravity

(ii) Size and shape features
- n: number of points on boundary
- ar: area
- per: perimeter
- p2a: p2a shape factor defined as perimeter squared divided by 4*pi*area
- np: number of relative maxima (number of corners) of the vector of radii (distance from center of gravity to points on boundary)

(iii) Orientation features
- alpha: orientation obtained as the angle between the horizontal and the closest corner (or maximum radius) in the clockwise direction
- angl: difference between a reference angle "gamma" and the orientation angle "alpha". "gamma" depends on the object shape and the assumed reference orientation

3.3 Part Recognition

As mentioned above, the objects we are dealing with are regular shapes and limited in number. For this case, a suitable method for part recognition is the use of a decision tree derived from prototypes of the parts to be recognized.

For this application the parts to be identified are chosen to be of the following shapes: circle, pentagon, triangle, square, or star. Any

other object would be considered as unknown and rejected. The decision tree technique was used to identify these shapes from the following features: p2a shape factor, number of points on boundary, and number of corners. The procedure produced the decision tree shown in Fig. 5.

Very good results are obtained using this technique, as long as the quality of the binary image is good. However, error rates increase as the image quality deteriorates.

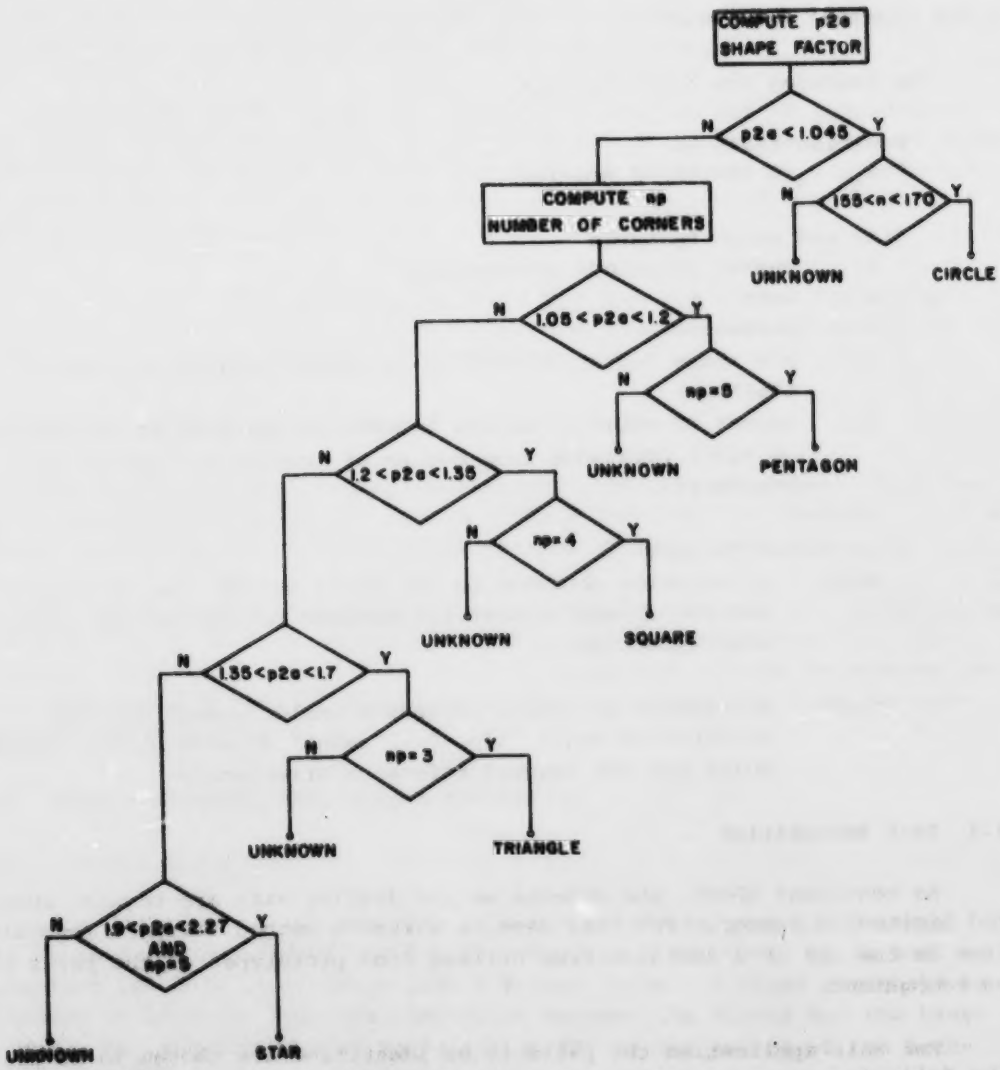

Fig. 5. The decision tree

3.4 Vision Program

The vision application program "fimage" is the main program responsible for part recognition and for the computation of its position and orientation. When the robot application program "fappl" requests a part name and pose, "fimage" takes an image, thresholds it, computes all the needed features of the first object found. After going through the decision tree, "fimage" computes the orientation of the part and returns its name, its center of gravity, its orientation to "fappl". The reader is referred to Chapter 6 for the details of intertask communications.

CHAPTER 4. VISION ALGORITHMS

In this chapter we explain in some detail the vision algorithms existing in the vision library. Equations and/or flowcharts will be given whenever needed. A listing of the routines can be found in Appendix E.

4.1 Thresholding

Subroutine name: "thresh"

Thresholding is a particularly useful technique for scenes containing solid objects resting on a contrasting background. The purpose is to segment the image into regions which can subsequently be analyzed based on their different features. Threshold selection involves choosing a grey level 't' such that all grey levels greater than 't' are mapped into the object label denoted by grey level '1', and all other levels are mapped into the background label denoted by grey level '0'.

In this work the threshold selection was done by trial and error. The value of the threshold grey level is held constant throughout the image since object and background grey levels are reasonably constant.

4.2 Edge Detection

4.2.1 Boundary Following Algorithm

Subroutine names: "bdrya", "bdryb"

The result of the thresholding process is a binary image with grey level '0' representing the background and grey level '1' representing the object. The boundary following algorithm starts by scanning the image until a '1' is found. It assumes that it has found a boundary and starts following it, changing every point belonging to the boundary to grey level '2'. The algorithm checks the neighborhood (4- or 8-neighborhood as specified by the user) of the pixel, to the outside of the boundary, until a

non-zero element is found. This is considered to be a new point on the boundary of the object. If no transition is found, this pixel is assumed to be noise (isolated point) and rejected. Scanning is then restarted where it stopped previously.

An object is considered found when two consecutive points on the boundary are identical to the first and second points. If the length of the boundary is less than a certain set limit it is rejected under the assumption that it is a small unknown object or noise. This limit is set by the user depending on his/her knowledge of the size of the objects and of the system parameters. This algorithm returns two arrays containing the x,y coordinates of the boundary of the object. Scanning then continues at the row of the image where it stopped previously to follow the contour.

The flowcharts of Figs. 6 and 7 show the details of this algorithm.

4.2.2 Thinning Algorithms

The following algorithms are used to smooth the boundary data.

The first routine "elextr" eliminates extra points from a list of contour points that form corners between two diagonal points in a 2 × 2 neighborhood.

The second routine "extr" eliminates extra points from a list of contour points that form a path normal to the contour in a 2 × 2 neighborhood.

The third routine "fold" eliminates any spikes from a list of contour points. A spike is a straight line of points, out of the normal direction of the contour, that would be included twice in the list because of folding.

4.3 Feature Extraction

Subroutine names: "acgp"
For more details, the reader is referred to reference 6.

4.3.1 Area

The area A is given by the formula

$$A = 1/2 \sum_{k=1}^{n-1} (x_k y_{k+1} - y_k x_{k+1})$$

where n is the number of points on the object contour. The last point on the contour list should be equal to the first, that is, $(x_1, y_1) = (x_n, y_n)$.

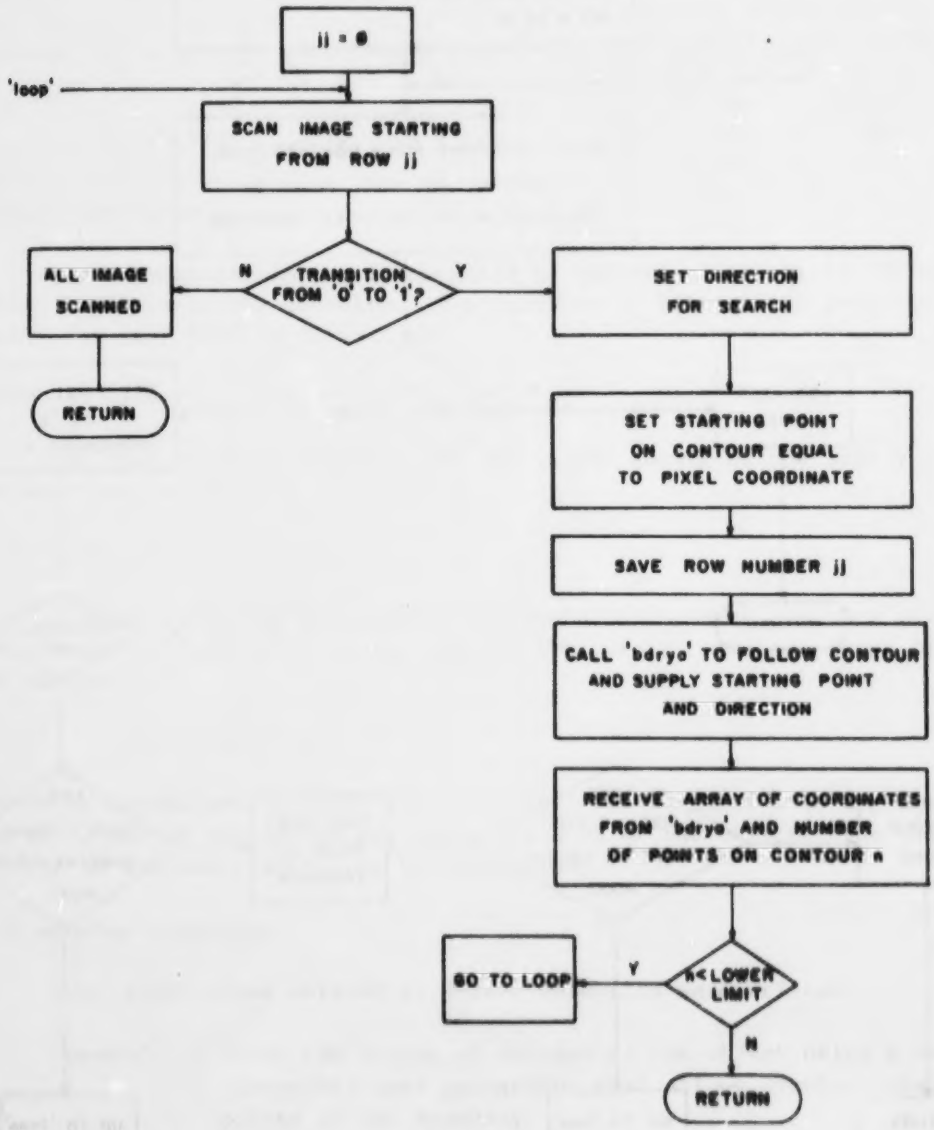

Fig. 6. Flowchart for boundary following 'bdryb'

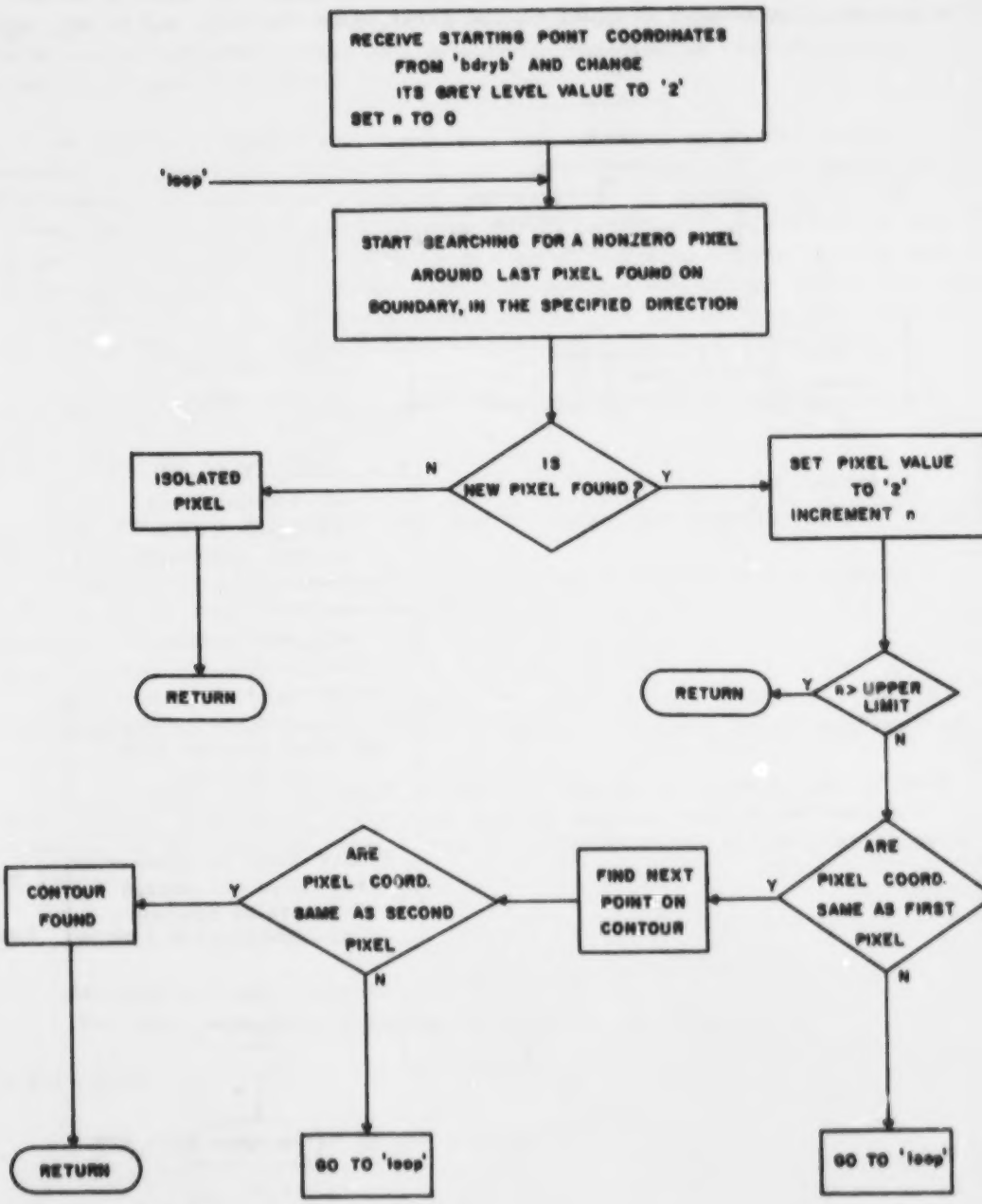

Fig. 7. Flowchart for boundary following 'bdrya'

4.3.2 Center of Gravity

The center of gravity (x_g, y_g) is computed using the following formulae:

$$x_g = 1/2 \sum_{k=1}^{n} x_k^2 (y_{k+1} - y_k)/A$$

$$y_g = -1/2 \sum_{k=1}^{n} y_k^2 (x_{k+1} - x_k)/A$$

4.3.3 Perimeter

The perimeter length is computed as the sum of the distances between every L points of the boundary. The distance d_i between the point i*L and the point (i - 1)*L is defined as:

$$d_i = ((x_{iL} - x_{(i-1)L})^2 + (y_{iL} - y_{(i-1)L})^2)^{1/2}$$

The last distance added to the sum is the length of the last j points in the list, where j is less than L.

4.3.4 The p2a Shape Factor

A shape factor is a dimensionless scale-independent quantity related to a predetermined ideal shape, usually the circle. The p2a shape factor is defined as

$$p2a = (per)^2/4\pi A$$

which is normalized to unity for a circle. The constant π is approximated by 22/7, per is the perimeter, and A the area of the object. The p2a factor is extremely sensitive to resolution.

4.4 Corner Detection

Four subroutines related to corner detection were written.

"peako" - finds the number of corners of the object using a method (described next paragraph) that allows overlap between points on the boundary
"peakn" - same as "peako" with no overlap
"peakad" - eliminates false corners, defined as being too close to the center of gravity
"peaksd" - eliminates false corners, defined as two peaks being too close to each other

The steps for corner detection are the following:

(i) Compute the average coordinate point (x, y) of L consecutive points on the contour. The segments of L points on a contour can be adjacent (peakn) or can overlap (peako).
(ii) Compute the distance between the average point and the center of gravity.
(iii) Repeat steps (i) and (ii) for all boundary points.
(iv) Find the number of corners by scanning the array of distances (radii) for the maxima. Special attention is given to both ends of the list to assure continuity.

Some of these corners may be false corners because of noisy data. Additional processing is done to eliminate a corner where the ratio of its radius to the maximum radius is less than a certain threshold (peakad). Also, if the distance between two corners is smaller than the smallest side of any of the blocks, the false corner is detected and removed.

4.5 Orientation

Subroutine names: "angor"

The computation of the orientation of an object depends on its shape. The orientation routine "angor" available in this library computes the angle between a point on the border of the object and the horizontal. The resulting angle is measured in the clockwise direction. The chosen point (x, y) on the boundary should represent some feature of the object. In our case, this point is chosen to be a maximum radius point, i.e., a corner.

Another routine "orient", computes a correction angle which is defined as the difference between a reference angle γ, that is constant for a part, and the orientation angle computed in "angor".

For example, the assumed reference orientation for the pentagon is as shown in Fig. 8a. Assuming that α is the angle of orientation of the

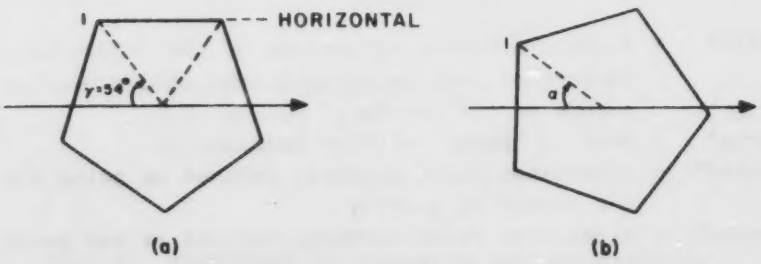

Fig. 8. Orientation for pentagon

pentagon when it is positioned randomly (Fig. 8b), then for the part to coincide with the reference position, the correction angle should be $(\gamma - \alpha)$.

CHAPTER 5. VISION CALIBRATION AND APPLICATION PROGRAM

This chapter outlines the robot task implementation. It is divided into two parts: first, calibration of the vision system and, second, the VAL program strategy to perform the assembly task. VAL (1) is not designed to support sensory control nor communicate with an external processor. This section describes our approach to working around these limitations. Familiarity with VAL is assumed.

5.1 Camera Calibration

Camera calibration is a procedure which allows one to relate coordinate measurements made by the vision system to the reference frame of the robot. In Fig. 9, the vectors "FOC" and "FOR" represent two independent descriptions of a frame F. The vector "FOC" represents the transformation to the origin of F from the camera base frame and vector "FOR" the transformation to the origin of F from the robot base frame. Although the blocks in our application are located by the camera in its coordinate system, grasping must be specified in the robot coordinate system. (We therefore require a description for the transformation "FOR" which uses the transformation "FOC".) From Fig. 9 the vector "FOR" is equivalent to the sum of the two vectors "COR" and FOC", or

$$FOR = COR + FOC$$

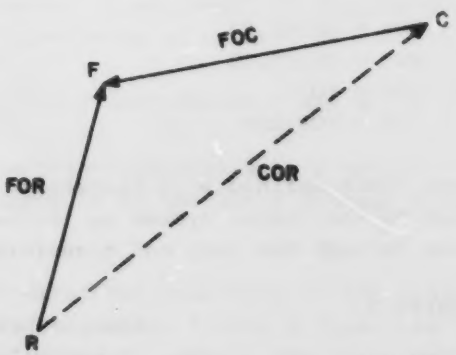

Fig. 9. Definition of camera frame referred to robot base frame

here "COR" represents the transformation to the camera base frame from the robot base frame.

Calibration is then the process of finding the transformation from the two known transformations "FOR" and "FOC" of a calibration through the expression

COR = FOR + COF

where COF is the inverse of FOC. The procedure outlined below uses the PUMA controller and VAL software to compute the desired transformations by defining the frames and performing the necessary matrix inversions.

The FRAME command in VAL defines a frame via three points, A0, the origin, A1, a point on the x-axis, and A2, a point in the x-y plane (distant from the x-axis for best accuracy). Two coincident frames "FOR" and "FOC" are described to the PUMA controller. Frame "FOR" is the robot secondary frame and is defined by "teaching" the controller the three points necessary for frame definition. These points are referred to as R0, R1, R2, and are measured with respect to the robot base frame. Frame "FOC" is the frame referenced to the camera, and the points (C0, C1, C2) necessary for its definition, and measured with respect to the origin of the camera field of view, are generated in the following way. Initially the three points for the vision frame definition are recorded in the PUMA controller as identical to those for the robot frame. These points are recorded by the vision system as they are taught to the robot controller. They are, however, recorded by the vision system in pixel coordinates. As these points have been recorded in both systems, we can calculate a vision scale factor (millimetres/ pixel). We then use the VAL monitor command "POINT" to enter the scaled x and y coordinates for the vision points previously labelled the controller. The following commands perform the required computations.

```
DO FRAME     FOR = R0,R1,R2
DO FRAME     FOC = C0,C1,C2
DO INVERSE   COF = FOC
DO SET       COR = FOR:COF
```

We now have the camera frame defined with respect to the robot base frame. A point "PICK" located by the vision system in its coordinates can now be specified to the robot through the compound transformation

GRASP = COR:PICK

5.2 VAL Application Program

The application program is written in the C language and runs on the Chorus processor. The detailed listing is in Appendix F. All the VAL

commands are executed in the monitor mode using the DO "VAL program step" command and are transmitted to the PUMA controller through the terminal input (Appendix A).

The program in essence moves the blocks between two taught positions. The destination is one of five sockets, each a taught location. The socket is selected after the matching block is identified by the vision software. The vision system also determines the position and orientation (pose) of the block to be grasped. The position and angle information are specified separately to the PUMA controller. The position data are transmitted to the controller and the gripper is moved to the grasp point. The gripper's orientation and the block's orientation from the vision system are used to specify angle correction data to the controller. The VAL commands required to perform these procedures are described in the following paragraphs.

Initially we specify to the VAL program a pick-up point called "PICK". Based on vision data, the coordinates of "PICK" are changed to correspond to the position of the next block to be grasped. The location of the block is specified by changing the x and y coordinates of the grasp point "PICK" with the "SHIFT" command.

In our demonstration, block orientation can be accommodated in the rotation of the PUMA joint 6. The learned position "PICK" is specified after an "ALIGN" command and as a result, the TOOL z-axis and the WORLD z-axis are aligned. Now joint 6 motion is pure rotation about the WORLD z-axis or block orientation. The joint 6 rotation has three components. The first is the "JOINT 6", the angle recorded when the position "PICK" was taught. As the arm is moved to the "shifted" position "PICK" the joint6 angle changes by an angle "incr.joint6". We therefore use the "APPRO" command to position the gripper above the block and then issue the "WHERE" command to find "incr.joint6". Finally, the vision system will provide "Ref_angle", which is the orientation angle referenced to the horizontal or base orientation (referred to as "angl" in Chapter 3). Therefore the final angle through which joint6 must be rotated is

joint6 = JOINT6 - incr.joint6 - Ref_angle

The "DRIVE" command to joint6 generates this motion. The "DRAW" command moves the suction gripper on to the block without a change in tool orientation.

The arm motion required for insertion of the grasped block in its proper socket has three segments; first, a departure from the tray sector specified by "DEPARTS" command; second, the large motion to bring the block above its socket provided by the "APPRO" command; and finally the "MOVE" command to insert the block into its socket. Figure 10 summarizes the complete sequence.

SHIFT	modify the x and y coordinates of "PICK" to the position coordinates of the block
APPRO	move suction gripper directly above the grasp point on the block
WHERE	find change in joint6 (incr.joint6) when the tool is moved to the "shifted" position of "PICK"
DRIVE	set joint6 angle so subsequent motion commands will result in correct orientation for insertion of the block into the socket joint6 = JOINT6 - incr.joint6 - Ref_angle
DRAW	position the gripper tip on the block
DEPARTS APPRO MOVE	move and insert the block

Fig. 10. VAL sequence for block grasping and insertion

CHAPTER 6. PUTTING IT ALL TOGETHER WITH INTERTASK COMMUNICATIONS

Intertask communication forms the framework upon which the entire system is coordinated. In this demonstration, two processors operate in parallel to generate robot control commands and to process video images. The software is divided into two corresponding tasks: the robot application task that resides on processor 0 and the vision task that resides on processor 1. These two tasks rely on intertask communications not only for data exchange but also for mutual synchronization.

In reality, a total of 11 tasks exists in this implementation. These include two Harmony-related tasks, six tasks for serial I/O, one notifier task that is required by the frame-grabber software (VGLIB), and the two tasks described above. This chapter is concerned with the role of intertask communications and in particular how it is applied in these latter two tasks.

Figure 11 is a skeletal view of the robot application task (task A) and the vision task (task V). The intertask communication is indicated by the arrows in the figure. Task A always begins with a fresh copy of task V by destroying and then re-creating it (lines 3A and 4A). After being activated from a user-entered command (line 5A), task A enters a loop and

Summary of Intertask Communication

Robot Application (task A)

```
                -
                -
1A    visionid = _Create( VISION );
                -
                -
2A  start:
3A    _Destroy( visionid );
4A    visionid = _Create( VISION );
                -
                -
5A  /* restart message to user */
                -
                -
6A    for( frame=1; - ; - )
      {         -
                -
7A      _Send( -,-, visionid );
8A      nbd = reply.STANDARD.RESULT;
9A      while( nbd != 0 )
        {
                -
                -
10A       _Send( -,-, visionid );
11A       nbd = reply.STANDARD.RESULT;
12A       if( nbd == 0 ) break;
                -
                -
    /* fetch and place part */
                -
                -
        }

      /* ready for next sector */

      }

      /* stop and reload the tray */

13A   goto start;
```

Vision Task (task V)

```
                -
                -
1V  top:
2V      req = _Receive( -,0 );
                -
                -
        /* take a picture and
           threshold it */
                -
                -
3V        nbd = -1;
                -
                -
4V      _Reply( -, req );
5V      nbd = 0;

6V  contour:
                -
      /* identify next part */
                -
                -
7V    req = _Receive( -,0 );
                -
      /* write contour and center
         of gravity to screen */
                -
                -
8V    _Reply( -, _Father_id() );
9V    if( nbd == 0 ) goto top;
10V   else goto contour;
```

Fig. 11. Summary of intertask communication

promptly sends a message to task V requesting it to take a picture and to report back when completed. At this point, task A becomes blocked and the task with the next highest priority on processor 0 becomes unblocked.

After being created, task V immediately becomes "receive blocked" (line 2V) and only unblocks after it has been sent a message. Upon receiving a message from task A, task V unblocks, takes a picture, displays it, thresholds it, and then displays the binary image. It replies to task A (line 4V) with a message that includes the value "nbd = -1". Task V continues to process the image and after identifying the first (next) object, it becomes "receive blocked" at line 7V.

When task A receives the _Reply from task V (line 7A), it unblocks and checks the value of nbd (line 9A). Task A is now ready for the coordinates, orientation, and identification of the first (next) object. On line 10A, task A sends a message to task V requesting these data. Upon receiving the message, task V unblocks (line 7V), writes the contour and the center of gravity to the monitor screen, replies to task A (line 8V) with the data, and then either branches to identify the next part or goes back to the beginning. If it branches to identify the next part, task V will again block at line 7V and await the message from task A. Both tasks will continue to loop in this manner until task V returns a value of nbd = 0. At this point task A will either branch back to the top of the "for loop" to process the next sector, or if all five sectors of the tray have been processed, it will branch to the very start to repeat the entire process.

CHAPTER 7. SUMMARY

This report describes a demonstration of a sensor-controlled robot using binary vision. Vision is a good example of the complex sensors which will become more common as robotics uses develop. These sensors will require high speed parallel processing of considerable amounts of data. Future applications will require a varying number of sensors all operating in parallel and essential for the coordinated, real time control of one or more robots.

The system described in this report meets these requirements and therefore should prove to be of continuing interest in the future.

ACKNOWLEDGEMENTS

The authors wish to thank Dr. W. Morven Gentleman who developed the Harmony operating system and its associated I/O servers and who provided guidance in this first application of Harmony.

They also wish to thank Paul Merrill of McGill University who provided the original software package that was used after some modification for remote computer control of the robot.

Finally the authors wish to acknowledge the technical support provided by Jim McDougall throughout this project.

REFERENCES

1. User's Guide To VAL (A Robot Programming and Control System) Version 12.4.6. Unimation Inc. Shelter Rock Lane, Danbury CT. May 1982.

2. D.A. Green. Chorus - A Multiprocessor Architecture For Real-Time Control Applications. NRC/ERB-964, National Research Council of Canada, Ottawa, Ont. December 1983.

3. Video Graphics Model VG-121/221 User's Manual. Datacube Inc., 4 Dearborn Road, Peabody, MA. August 1982.

4. W.M. Gentleman. Using the Harmony Operating System. NRC/ERB-966, National Research Council of Canada, Ottawa, Ont. December 1983.

5. C Programmer's Manual. Whitesmiths Ltd. 97 Lowell Road, Concord MA. November 1981.

6. R. Horaud, S. Olympieff, and J.P. Charras. Shape and Position Recognition of Mechanical Parts from their Outlines. 1st Int. Conf. on Robot Vision and Sensory Control. 1981. pp. 125-134.

APPENDIX A

Listing of VALPAK, VALLO, Getfloat

```
/* VALPAK */

extern float Getfloat ();
extern int get_msno ();
extern send_rb();
extern wait_for ();

static char wher_str[] = "W";
static char null_str[] = "";
static char algn_str[] = "DO AL";

static char in_str[250]=0; /* array used by all routines to get the robot
                                    controller's response                */

struct UCB *ucb_iv = 0 , *ucb_ov = 0 ;
static struct UCB *old_ucbi = 0, *old_ucbo = 0;

delay (time, message)   /* val delay command */

char message [];
int time;
{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO DELAY %i", time);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}


speed (value, message)
```

```
/* Sets the robot speed. Maximum speed allowed is 100.
                        If the specified value is less than 0 or greater than
                        100, a speed of 0 or 100 is send.
                        A pointer to the integer value of speed is passed as
                        as a parameter for this routine                    */
int value;
char message [];

{
        int response;

        if (value > 100)
                value = 100;


        if (value < 0)
                value = 0;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "SPEED %i", value);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}


set (ltr_num, rtr_num, message)

        /* val SET command */

int ltr_num, rtr_num;
char message [];
        {

        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO SET TRANZi = TRANZi", ltr_num, rtr_num);
```

```
        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
        }


setcl (ltr_num, rtr_num1, rtr_num2, message)

        /* val SET command for 1st level compound transformations */

int ltr_num, rtr_num1, rtr_num2;
char message [];

        {

        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO SET TRAN%i = TRAN%i:TRAN%i",
                ltr_num, rtr_num1, rtr_num2);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
        }


drive (joint, change, sped, message)

/*      change: a pointer to a floating point number that
                specifies the change in degrees */
```

```
int joint, sped;
float change;
char message [];

{

        int response;

        if (joint > 0 && joint < 7 && sped > 0 && sped <= 100)
           {
                fill (in_str, 30, '\0');
                decode (in_str, 30, "DO DRI %i, %.3f, %i", joint,
                        change, sped);

                old_ucbi = _Selectinput ( ucb_iv );
                old_ucbo = _Selectoutput ( ucb_ov );

                send_rb (in_str);   /* transmit command string */
                response = get_msno ( message );

                _Selectinput ( old_ucbi );
                _Selectoutput ( old_ucbo );

                return ( response );
           }
        else return (100);
}


depart (distance, message)
/* move the robot tool the current Z axis of the tool */

float distance;
char message [];

{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO DEPART %.3f", distance);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );
```

```
        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );

}

departs (distance, message)
/* Move the robot tool along the current Z axis of the tool.
                    The motion is along a straight line.                    */
float distance;
char message [];

{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO DEPARTS %.3f", distance);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}

signal (sig_num,message)

/*  turns signal on/off,  the pointer sig_num points to a value that specifies
        the channel; a positive number turns the channel on, negative - off */

int sig_num;
char message[];
{
        int response;

        if(sig_num > -8 && sig_num < 8 && sig_num != 0)
        {
```

```
                fill (in_str, 50, '\0');
                decode (in_str, 50, "DO SI %i", sig_num);

                old_ucbi = _Selectinput ( ucb_iv );
                old_ucbo = _Selectoutput ( ucb_ov );

                send_rb (in_str);   /* transmit command string */
                response = get_msno ( message );

                _Selectinput ( old_ucbi );
                _Selectoutput ( old_ucbo );

                return ( response );
        }
        else                        /* error in parameters */
        return(100);
}


shift (tran_num, ddx, ddy, ddz, message)

/* VAL SHIFT<transformation> BY <dx>,<dy>,<dz> unlike val all three coordinat
                arguments must be specified  */

char message[];
int tran_num;
float ddx,ddy,ddz;
{
        int response;

        fill (in_str, 50, '\0');
decode (in_str,50,"DO SH TRAN%i BY %.3f, %.3f, %.3f",tran_num,ddx,ddy,ddz);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}
```

```
draw (dx, dy, dz, message)
                        /* val draw command */
float dx, dy, dz;
char message [];

{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO DRA %.3f, %.3f, %.3f", dx,dy,dz);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );
        return ( response );
}



align (message)

char message[];
{

        int response;

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (algn_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}



where ( xc, yc, zc, oc, ac, tc, jt1, jt2, jt3, jt4, jt5, jt6 )
```

```
float *xc, *yc, *zc, *oc, *ac, *tc, *jt1, *jt2, *jt3, *jt4, *jt5, *jt6;

{
        char *ptr;

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (wher_str);   /* transmit command string */

        get_str (in_str);
        /* get the second & third line that the LSI transmits */

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        ptr = in_str;
        *xc = Getfloat(&ptr);
        *yc = Getfloat(&ptr);
        *zc = Getfloat(&ptr);
        *oc = Getfloat(&ptr);
        *ac = Getfloat(&ptr);
        *tc = Getfloat(&ptr);
        *jt1 = Getfloat(&ptr);
        *jt2 = Getfloat(&ptr);
        *jt3 = Getfloat(&ptr);
        *jt4 = Getfloat(&ptr);
        *jt5 = Getfloat(&ptr);
        *jt6 = Getfloat(&ptr);

        return (62);                    /* return ok */
}


move (location, message) /* moves the robot to the location
                            and orientation specified by the transformation
                            number pointed by the parameter location.
                            The motion is Joint-interpolated */

int location;
char message [];


{
        int response;
```

```
        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO MOVE TRANZi", location);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}


moves (location, message)
/* Moves the robot to the location orientation specified
                        by the transformation pointed by the parameter
                        location. Only the transformation number is passed.
                        The routine moves the tool along a straight line */

int location;
char message [];
{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO MOVES TRANZi", location);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}
```

```
appro (location, distance, message)

/* Moves the tool to the position and orientation defined by
                the transformation number pointed by the variable location,
                and an offset (positive or negative) along the tool z axis,
                of the distance (in mm) pointed by the parameter distance.
                The resulted motion is joint-interpolated                    */

int location;
float distance;
char message [];

{
        int response;

        fill (in_str, 50, '\0');
        decode (in_str, 50, "DO APPRO TRANX%i,%f", location, distance);

        old_ucbi = _Selectinput ( ucb_iv );
        old_ucbo = _Selectoutput ( ucb_ov );

        send_rb (in_str);   /* transmit command string */
        response = get_msno ( message );

        _Selectinput ( old_ucbi );
        _Selectoutput ( old_ucbo );

        return ( response );
}


appros (location, distance, message)

/* Moves the tool to the position and orientation defined by
                the transformation number pointed by the variable location,
                and an offset (positive or negative) along the tool z axis,
                of the distance (in mm) pointed by the parameter distance.
                The routine moves the tool along a straight line             */

int location;
float distance;
char message [];

{
        int response;
```

```
    fill (in_str, 50, '\0');
    decode (in_str, 50, "DO APPROS TRAN%i,%f", location, distance);

    old_ucbi = _Selectinput ( ucb_iv );
    old_ucbo = _Selectoutput ( ucb_ov );

    send_rb (in_str);   /* transmit command string */
    response = get_msno ( message );

    _Selectinput ( old_ucbi );
    _Selectoutput ( old_ucbo );

    return ( response );
}
```

```
                                        /* VALLO */

#include "[harmony.src]sys.h"

#define FAILURE 100             /* default message number */
#define SIZE (sizeof(rb_msgs) / sizeof(struct response))

static char pr_ch={0};

send_rb (string)            /* Transmits a character string to the robot
                               controller. The string should be properly
                               terminated with a NULL character. The command
                               is terminated with a CR which is NOT part of
                               the string.                                */

char string[];

{
        int i;

        for (i = 0; string[i]; i++)
        {
                pr_ch = string[i];
                _Put(pr_ch);
        }
        _Put('\n');       /*  CR to terminate the command */
        _Flush ();
}

struct response {
        char *string;     /* minimum string required to identify the message *
        int  rtn_num; /* message_number returned to the users program */
}
rb_msgs [] = {
        "*\"$", 0,
        "*AM$", 1,
        "*ARI$", 2,
        "*ARM$", 3,
        "*BAD A$", 4,
        "*BAD C$", 5,
        "*BAD P$", 6,
        "*BAD S$", 7,
        "*BR$", 8,
        "*CAN'T GO A$", 9,
        "*CAN'T GO O$", 10,
        "*CAN'T I$", 11,
```

```
"*CH$", 12,
"*CL$", 13,
"*COMM$", 14,
"*COMP$", 15,
"*DIR$", 16,
"*DIS$", 17,
"*DUP$", 18,
"*FILE ALREADY E$", 19,
"*FILE ALREADY O$", 20,
"*FILE F$", 21,
"*FL$", 22,
"*FU$", 23,
"*HAN$", 24,
"*HAR$", 25,
"*ILLEGAL C$", 26,
"*ILLEGAL F$", 27,
"*ILLEGAL I$", 28,
"*ILLEGAL J$", 29,
"*INI$", 30,
"*INPUT B$", 31,
"*INPUT E$", 32,
"*INVALID L$", 33,
"*INVALID N$", 34,
"*INVALID S$", 35,
"*L$", 36,
"*ME$", 37,
"*MI$", 38,
"*NO PROGRAM SP$", 39,
"*NO PROGRAM ST$", 40,
"*NO Z$", 41,
"*NON$", 42,
"*NOT A$", 43,
"*NOT C$", 44,
"*PA$", 45,
"*PR$", 46,
"*STORAGE A$", 47,
"*STORAGE S$", 48,
"*STR$", 49,
"*T$", 50,
"*UND$", 51,
"*UNE$", 52,
"*V$", 53,
"*W$", 54,
"*[FATAL] A$", 55,
"*[FATAL] C$", 56,
"*[FATAL] E$", 57,
```

```
        "*[FATAL] L$", 58,
        "*[FATAL] O$", 59,
        "*[FATAL] SERVO D$", 60,
        "*[FATAL] SERVO R$", 61,
        ".$", 62,
        "AB$", 63,
        "AL$", 72,    /* VAL staring message, V lost as LF.... */
        "AR$", 64,
        "E$", 65,
        "H$", 66,
        "I$", 67,
        "L$", 68,
        "O$", 71,
        "PA$", 69,
        "PR$", 70,
        "V$", 72,
        "$" , 100
};
get_mes (message)

char message [];

{
        int end = 1, mes_num, chr_num, i;
        int msg_num = 0, limit;
        char c, r_char;

        limit = SIZE -1;
        i = 0;
        for (mes_num = 0, chr_num = 0; end != 0; )
        {
                r_char = _Get();        /*  skip LF */
                if (  r_char == '\12' ) r_char = _Get();
                message [i++] = r_char;

                while (
                mes_num < SIZE
                    && limit > mes_num
                    && chr_num < count (rb_msgs[mes_num].string)
                    && r_char > *(rb_msgs[mes_num].string + chr_num)
                    )
                        mes_num++;

                if (mes_num >= SIZE
                || limit < mes_num
                || chr_num >= count (rb_msgs[mes_num].string)
```

```
                ;; r_char < *(rb_msgs[mes_num].string + chr_num)
                )
                {
                        msg_num = FAILURE;
                        end = 0;
                }
                else    /* if two characters match..... */
                {
                        limit = w_end (rb_msgs, r_char, mes_num,
                                        chr_num, limit);
                        /* string window */
                        chr_num++;

                        if ((c = *(rb_msgs[mes_num].string +
                                  chr_num)) == '$'
                            ;; c == '\0')
                        {
                                msg_num =
                                      rb_msgs[mes_num].rtn_num;
                                end = 0;
                        }
                }   /* otherwise continue in the loop */
        }
        message [i] = '\0';
        return (msg_num);
}

count (pointer) /* Function similar to sizeof(). It returns the size of the */
/* array pointed by the arg. pointer                                        */

char *pointer;

{
        int i = 0;
        while (*(pointer + i))
                i++;
        return (i);
}

w_end (look, ch, first, number, limit) /* Checks vertically the structure and
                                  identifies the end of the current "window" and
                                  returns the number of the last line that could
                                  match the current message                  */

struct response look[];
char ch;
```

```
int first, number, limit;

{
        int last;

        last = first;
        while (*(look[last].string + number) == ch && last < limit)
                last++;

        return (last);
}

wait_for (ch, string)   /* waits until ch is found
                           all read in characters are put in string  */

char ch, string[];

{
        int i;

        for (i = 0; ( string[i++] = _Get() ) != ch ; )

        string[i] = '\0';
}
get_msno (in_str)

char in_str[];

{
        int msg_num;
        char temp1 [80], temp2 [80], temp3 [80];

        msg_num = get_mes (temp1);

        switch (msg_num) {

        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 7:
        case 8:
        case 9:
```

```
case 10:
case 11:
case 12:
case 13:
case 14:
case 15:
case 16:
case 17:
case 18:
case 19:
case 20:
case 21:
case 22:
case 24:
case 25:
case 26:
case 27:
case 28:
case 29:
case 30:
case 31:
case 32:
case 33:
case 34:
case 35:
case 36:
case 38:
case 39:
case 40:
case 42:
case 43:
case 45:
case 47:
case 48:
case 49:
case 50:
case 51:
case 52:
case 53:
case 54:
case 63:
case 65:
case 68:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
```

```
case 6:
case 23:
case 41:
case 55:
case 57:
case 58:
case 59:
case 60:
case 61:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 46:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 37:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 56:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 66:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 69:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 70:
        wait_for ('.', temp2);
        fill (in_str, 100, '\0');
        decode (in_str, 100, "%p%p", temp1, temp2);
        return (msg_num);
case 44:
case 64:
```

```
        case 67:
                wait_for ('?', temp2);
                wait_for (' ', temp3);
                fill (in_str, 180, '\0');
                decode (in_str, 180, "%p%p%p", temp1, temp2, temp3);
                return (msg_num);
        case 62:                        /* everything ok */
                in_str [0] = '\0';
                return (msg_num);
        case 71:                        /* message is OK (after calibration) */
                wait_for ('.', temp2);
                in_str [0] = '\0';
                return (msg_num);
        case 72:
                wait_for ('?', temp2);
                wait_for (' ', temp3);
                fill (in_str, 180, '\0');
                decode (in_str, 180, "%p%p%p", temp1, temp2, temp3);
                return (msg_num);
        default:                        /* unknown error message */

                wait_for ('.', temp2);
                in_str [0] = '\0';
                return (msg_num);
        }

}

get_str (char_str)  /* used by the WHERE routine to get the LSI response */

char char_str[];
{
        int i = 0, count = 0, space = 0;
        char rch;

        while (count < 5)  /* after the 5th LF terminate */
        {

                rch = _Get();

                if (rch == '\12')       /* if LF       */
                {
                        space = 1;
                        count++;                /* count line feed  */

                        if (count == 4)
                                char_str[i++] = ' ';
```

```
                }
                else
                {
                        if (count == 2 || count == 4)
                                if (rch == ' ' && space == 0)
                                {
                                        char_str[i++] = rch;
                                        space++;
                                }
                                else if (rch != ' ')
                                {
                                        space = 0;
                                        char_str[i++] = rch;
                                }
                }
        }
        char_str[i] = '\0';

/*      wait for '.' response   */

        while ( ( rch = _Get() ) != '.' )
                ;               /*  do nothing  */
}
```

```
/* Getfloat.c */

/* convert string to floating point */


float   Getfloat (s)
        char **s;

{
        int sign = 1;
        float num = 0, frac = 0, dig = 1;

          for (;;)

             {
                while (**s == ' ') (*s)++;

                   for (; **s != '.'; (*s)++)
                        {
                           if (**s == '-') sign = -1;
                           else if ('0' <= **s && **s <= '9')
                                {
                                   **s -= '0';
                                   num = (num * 10) + **s;
                                }
                           else {
        _Putstr ("getfloat returns 0\n");
        _Flush ();
                                        return (0);     /*      bad string */
                                }
                        }
                   (*s)++;

                   for (; **s != ' ' && **s != '\0'; (*s)++)

                        {
                           if ('0' <= **s && **s <= '9')
                                {
                                   **s -= '0';
                                   dig *= 10;
                                   frac = frac + (**s / dig);
                                }
                        }
                (*s)++;
                return (sign * (num + frac));
             }
          }
```

APPENDIX B

Listing of Ehvt_server

```
typedef struct EHVT_STATE
  {
    unsigned EHVTIN_REQUESTOR, EHVTOUT_REQUESTOR;
    short int EHVTIN_COUNT;
    char EHVTOUT_COUNT, EHVT_WAIT_ECHO;
    char EHVTIN_LINE[320], EHVTOUT_LINE[20];
  };

typedef union EHVT_RQST
  {
    struct STD_RQST                    STANDARD;
    struct
      {
        union BUFF_STRUCT              BUFF_HDR;
        char                           EHVT_BUFF_TEXT[20];
      } EHVT_USR_RQST;
    struct TTY_INPUT                   FROM_TTY;
    struct REPORT_SERVER               TTY_REPORT;
    struct DEVICE_IDENTIFICATION       ACIA_IDENTIFICATION;
    struct OPEN_RQST                   TTY_OPEN;
    struct CLOSE_RQST                  TTY_CLOSE;
  };

_Ehvt_server()
    /* Copyright National Research Council of Canada, 1983 */
    /* root function for echo handshake virtual terminal server                */
    /* this server appears as a terminal to a remote computer connected to     */
    /* this ACIA. No characters sent from the remote computer are echoed       */
    /* back to it. Flow control to avoid overrunning the remote computer is    */
    /* achieved by waiting for the echo from each character sent to it         */
    /* before sending the next. Echo characters and other uninteresting        */
    /* characters are discarded from the input stream that this server         */
    /* makes available to other tasks.                                         */
    /* the ACIA controlled by this server is assumed to be ACIA 1 on           */
    /* processor 1, so EHVTO and EHVTI must be for task templates on           */
    /* _Pnumber 1                                                              */
  {
    unsigned tti, tto, requestor;
    union EHVT_RQST request;
    struct STD_RPLY std_rply;
    struct OPEN_RPLY open_rply;
    struct EHVT_STATE ehvt_state;
```

```
struct TTY_OUTPUT out_msg;
register short int i, lim, cons;
short int prod;
char c;

extern unsigned *_Loc_directory_id;
extern char _ACIA1_cntrl;

/* Initialize */
tti = _Create( EHVTI );
tto = _Create( EHVTO );
ehvt_state.EHVTIN_COUNT = 0;
ehvt_state.EHVTOUT_COUNT = 0;
ehvt_state.EHVT_WAIT_ECHO = 0;
ehvt_state.EHVTIN_REQUESTOR = 0;
ehvt_state.EHVTOUT_REQUESTOR = 0;
prod = 0;
cons = 0;

request.STANDARD.MSG_SIZE = sizeof( struct DEVICE_IDENTIFICATION );
request.ACIA_IDENTIFICATION.DEVICE_CODE = RECV_1ACIA;
request.ACIA_IDENTIFICATION.DEVICE_ADDRESS = (char *)0xFFFF20;
request.ACIA_IDENTIFICATION.CNTRL_COPY = &_ACIA1_cntrl;
std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
_Send( (char *)&request, (char *)&std_rply, tti );

request.STANDARD.MSG_SIZE = sizeof( struct DEVICE_IDENTIFICATION );
request.ACIA_IDENTIFICATION.DEVICE_CODE = XMIT_1ACIA;
request.ACIA_IDENTIFICATION.DEVICE_ADDRESS = (char *)0xFFFF20;
request.ACIA_IDENTIFICATION.CNTRL_COPY = &_ACIA1_cntrl;
std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
_Send( (char *)&request, (char *)&std_rply, tto );

request.STANDARD.MSG_SIZE = sizeof( struct REPORT_SERVER );
request.STANDARD.MSG_TYPE = REPORT_FOR_SERVICE;
for( i=0;
    request.TTY_REPORT.NAME_REPORTED[i] = "EHVT:"[i];
    ++i ) ;
std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
_Send( (char *)&request, (char *)&std_rply, *_Loc_directory_id );

for(;;)
  {
    request.STANDARD.MSG_SIZE = sizeof( union TTY_RQST );
    requestor = _Receive( (char *)&request, 0 );
```

```
switch( request.STANDARD.MSG_TYPE )
  {
    case READ_REQUEST:
      {
        ehvt_state.EHVTIN_REQUESTOR = requestor;
        break;
      }
    case WRITE_REQUEST:
      {
        ehvt_state.EHVTOUT_REQUESTOR = requestor;
        for( i=request.TTY_USR_RQST.OUT_BUFFER.OUT_LENGTH; i;)
            ehvt_state.EHVTOUT_LINE[ehvt_state.EHVTOUT_COUNT++]
                = request.TTY_USR_RQST.OUT_BUFFER.OUT_DATA[--i];
        break;
      }
    case FLUSH_REQUEST:
      {
        /* not right, but fix later */
        std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
        _Reply( (char *)&std_rply, requestor );
        break;
      }
    case TTO_AVAILABLE:
      {
        tto = requestor;
        break;
      }
    case BYTE_FROM_TTI:
      {
        std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
        _Reply( (char *)&std_rply, requestor );
        c = request.FROM_TTY.IN_TTY_BYTE;
        if( ehvt_state.EHVT_WAIT_ECHO && ( c == ehvt_state.EHVT_WAIT_ECHO ) )
          {
            ehvt_state.EHVT_WAIT_ECHO = 0;
            if( !ehvt_state.EHVTOUT_COUNT )
              {
                std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
                _Reply( (char *)&std_rply, ehvt_state.EHVTOUT_REQUESTOR );
                ehvt_state.EHVTOUT_REQUESTOR = 0;
              }
          }
        else if( _Ehvt_interesting( c )
            && (ehvt_state.EHVTIN_COUNT++ < 320) )
```

```
            {
              ehvt_state.EHVTIN_LINE[prod++] = c;
              if( prod = 320 ) prod = 0;
            }
          break;
        }
      case OPEN_REQUEST:
        {
          /* for now, we will not record connection */
          open_rply.STANDARD.MSG_SIZE = sizeof( struct OPEN_RPLY );
          open_rply.OPEN_FAILURE_CODE = 0;
          open_rply.OPEN_CONNECTION = 1;  /* arbitrary */
          open_rply.OPEN_SERVER = _My_id();
          if( request.TTY_OPEN.MODE & READ )
              open_rply.RQD_BUFF_SIZE = 10;
          else open_rply.RQD_BUFF_SIZE = 20;
          open_rply.OPEN_FLAGS = 0;

          _Reply( (char *)&open_rply, requestor );
          break;
        }
      case CLOSE_REQUEST:
        {
          /* for now, as connection not recorded,just reply */
          std_rply.MSG_SIZE = sizeof( struct STD_RPLY );
          _Reply( (char *)&std_rply, requestor );
          break;
        }
    }

  if( ehvt_state.EHVTIN_REQUESTOR && ehvt_state.EHVTIN_COUNT > 0 )
    {
      lim = ehvt_state.EHVTIN_COUNT;
      if( lim > 10 ) lim = 10;
      ehvt_state.EHVTIN_COUNT -= lim;
      request.IN_BUFFER.IN_LENGTH = lim;
      request.IN_BUFFER.MSG_SIZE = sizeof( union BUFF_STRUCT ) - 1 + lim;
      for( i=0; i<lim; ++i )
        {
          request.IN_BUFFER.IN_DATA[i] = ehvt_state.EHVTIN_LINE[cons++];
          if( cons = 320 ) cons = 0;
        }
      _Reply( (char *)&request, ehvt_state.EHVTIN_REQUESTOR );
      ehvt_state.EHVTIN_REQUESTOR = 0;
    }
```

```
        if( tto && ehvt_state.EHVTOUT_COUNT
            && !ehvt_state.EHVT_WAIT_ECHO )
          {
            out_msg.MSG_SIZE = sizeof( short int ) + sizeof( char ) + sizeof( char );
            out_msg.TTO_COUNT = 1;
            out_msg.OUT_TTY_BYTES[0] = ehvt_state.EHVT_WAIT_ECHO
                = ehvt_state.EHVTOUT_LINE[--ehvt_state.EHVTOUT_COUNT];
            if( out_msg.OUT_TTY_BYTES[0] == '\n' )
                out_msg.OUT_TTY_BYTES[0] = ehvt_state.EHVT_WAIT_ECHO = '\r';
            _Reply( (char *)&out_msg, tto );
            tto = 0;
          }
      }
  };

_Ehvt_interesting( c )
    /* Copyright National Research Council of Canada, 1983 */
    char c;
  {
    if( c > 126 ) return( 0 );          /* not DEL, no bytes with MS bit on */
    if( c == '\n' ) return( c );        /* LF */
    if( c < 0x20 ) return( 0 );         /* no other control characters */
    return( c );
  };
```

APPENDIX C

Listing of VGLIB

ACFIELD.S

```
*
*
*
*       C compatible subroutine
*       Digitizes a single even field on the vg-121
*       No arguments are required.
*       Interrupt driven version prepared 12 August 1983
*
*       NOTE: this routine requires a "notifier task"
*       that runs at Harmony  priority level 4 ( the vg121 h/w
*       interrupt level ). This routine will "receive block"
*       until the notifier receives the vg121 interrupt and
*       sends a message.
*
*
                .globl  _acfield
                .text
*
        _acfield:
                link    a6, #0
                movea.l #0xfdff70, a0           * vg-121 pointer
*
                move.b  #0x24, 1(a0)            * issue command
                move.b  #0x20, 1(a0)            * drop command
*
                move.w  #2, rqstbfr             * msg length
                pea     0                       * push args
                pea     rqstbfr
                jsr     __Receive               * wait for notifier
                addq.l  #8, sp                  * restore sp
                unlk    a6
                rts
*
                .data
*
         rqstbfr:
                .byte   0, 0
*
                .globl  __Receive
```

PIXPOS.S

```
*
*
*       C compatible subroutine.
*       (No Interrupt handling required)
*       loads vg-121 horizontal and vertical position registers.
*       2 longword arguments are passed via the stack as:
*               top of stack:   horizontal (pixel) position (0-319)
*               next word:      vertical (line) position (0-1023)
*
*               i.e.    _pixpos(hpos,vpos)
*
*
                .text
*
                .globl  _pixpos
*
        _pixpos:
                link    a6,#0
                move.l  d3, -(sp)       * save d3
                movea.l #0xfdff70, a0   * vg-121 pointer
*
                move.l  8(a6), d3       * get 1st arg (h pos)
                move.b  d3, 4(a0)       * load horiz low
                lsr.w   #8, d3          * shift
                move.b  d3, 5(a0)       * load horiz high
*
                move.l  12(a6), d3      * get 2nd arg (v pos)
                move.b  d3, 6(a0)       * load vert low bits
                lsr.w   #8, d3          * shift
                move.b  d3, 7(a0)       * load vert high bits
*
                move.l  (sp)+, d3       * restore d3
                unlk    a6
                rts
```

```
*                                   RDFIELD.S
*
*
*
*       C compatible subroutine.
*       Interrupt driven version.
*       Transfers a complete vg-121 field starting at pixel position
*       specified by the vertical and horizontal position registers.
*       Pixels are transferred in byte sequence to system memory
*       beginning at address pointed to by the argument.
*       1 longword argument is required.
*
*
                .text
*
                .globl  _rdfield
*
        _rdfield:
                link    a6, #0
                movea.l #0xfdff70, a0           * vg-121 pointer
                movea.l 8(a6), a1               * get arg (dest pointer)
                move.l  #77439, d1              * pixel counter
                move.b  #0x01, 1(a0)            * autoincr read
*
        2:      move.b  (a0), (a1)+             * read pixel
                subq.l  #1, d1                  * decr pixel counter
                bne     2b                      * loop
*
*       all done
*
                unlk    a6
                rts
```

```
*                                   SYNCMODE.S
*
*
*       C compatible subroutine.
*       Prepared 25 April 1983
*
*       interrupt driven version 12 August 1983
*
*       Sets video synchronization mode for vg-121.
*       one argument is required.
*
*                       syncmode( n )
*
*                       n = 0   crystal sync
*                       n = 1   phase lock to video input
*                       n = 2   phase lock to external sync
*                       n = 3   phase lock to external drive
*
                .text
                .globl  _syncmode
*
        _syncmode:
                link    a6, #0
                movea.l #0xfdff70, a0           * vg-121 pointer
*
                move.l  8(a6), d0               * get arg
                lsl.w   #4, d0                  * shift bits
                move.b  3(a0), d1               * get startline high
                andi.b  #0x0f, d1               * mask position bits
                or.b    d0, d1                  * add sync mode
                move.b  d1, 3(a0)               * write sync mode
*
                unlk    a6
                rts
```

WASH.S

```
*
*
*       C compatible subroutine clears the vg-121 screen to black (0x0).
*       No arguments are required.
*
*       Interrupt driven version 12 August 1983
*
*       This routine requires the use of a "notifier task"
*       that runs at the same Harmony priority level as the VG121
*       ( level 4 ). The routine will "receive block" until the
*       notifier sends it a message. This only occurs after the notifier
*       receives an interrupt from the VG121.
*
                .text
*
                .globl  _wash
*
        _wash:
                link    a6,#0
                movea.l #0xfdff70, a0   * vg-121 pointer
*
                move.b  #0, (a0)        * black
                move.b  #0x25, 1(a0)    * blacken screen
                move.b  #0x20, 1(a0)    * drop command
*
                move.w  #2, rqstbfr     * msg length
                pea     0               * push args
                pea     rqstbfr
                jsr     __Receive       * wait for msg from taxi
                addq.l  #8, sp          * restore sp
*
                unlk    a6              * remove stack frame
                rts
*
                .data
*
        rqstbfr:
                .byte   0, 0
*
                .globl  __Receive
```

WINDOW.S

```
*
*
*       C compatible subroutine
*       (no interrupt handling required)
*       loads vg-121 starting line register
*       ( video synchronization mode is unaffected ).
*       1 longword argument is passed as:
*                       _window( arg )
*
*
                .text
*
                .globl  _window
*
        _window:
                link    a6, #0
                movea.l #0xfdff70, a0           * vg-121 pointer
*
                move.l  8(a6), d0               * get argument
                move.b  d0, 2(a0)               * load startline low
                lsr.w   #8, d0
                andi.b  #0x03, d0               * 2 bits please !
                move.b  3(a0), d1               * get startline high
                andi.b  #0xfc, d1               * mask sync mode bits
                or.b    d1, d0                  * add in position bits
                move.b  d0, 3(a0)               * load startline high
*
                unlk    a6
                rts
```

WRFIELD.S

```
*
*
*
*       c compatible subroutine.
*       Interrupt driven version 12 August 1983
*       Writes a complete field to the vg-121 via autoincrementing
*       starting at pixel position specified by the vertical and
*       horizontal position registers.
*       Pixels are transferred in byte sequence from system memory
*       beginning at address pointed to by argument.
*       1 longword argument is required.
*
*

        .text
*
        .globl  _wrfield
*
_wrfield:
        link    a6, #0
        movea.l #0xfdff70, a0           * vg-121 pointer
        movea.l 8(a6), a1               * get arg ( src pntr )
        move.l  #77439, d1              * pixel counter
        move.b  #0x02, 1(a0)            * autoincr on write
*
1:      move.b  (a1)+, (a0)             * write pixel
        subq.l  #1, d1                  * decr pixel cntr
        bne     1b
*
        unlk    a6                      * done
        rts
```

WRPIXEL.S

```
*
*
*
*       C compatible subroutine.
*       Interrupt driven version 12 August 1983
*       Writes a single pixel to the vg-121.
*       The vertical and horizontal positions and the intensity
*       value are passed as arguments.
*       3 longwords are required. (the Whitesmith's C compiler
*       converts chars to long ints).
*
*               wrpixel( hpos, vpos, inten )
*
*
                .text
                .globl  _wrpixel
*
        _wrpixel:
                link    a6, #0
                movea.l #0xfdff70, a0           * vg-121 pointer
*
*       horizontal position register
*
                move.l  8(a6), d0               * get 1st arg ( h pos )
                move.b  d0, 4(a0)               * load horiz low
                lsr.w   #8, d0                  * shift
                move.b  d0, 5(a0)               * load horiz high
*
*       vertical position register
*
                move.l  12(a6), d0              * get 2nd arg ( v pos )
                move.b  d0, 6(a0)               * load vert low
                lsr.w   #8, d0
                move.b  d0, 7(a0)               * load vert high
*
*       write pixel
*
                move.l  16(a6), d0              * get pixel value
                move.b  #0x0, 1(a0)             * vg-121 register mode
                move.b  d0, (a0)                * write pixel
*
                unlk    a6                      * bail out
                rts
```

APPENDIX D

Listing of Vision Application Program

/* defin.h */

```
/* The include file for vision          */


#define   COL    319
#define   ROW    241
#define   THD    0x3d
#define   MAXN   100
#define   NC     8
#define   LENG   450
#define   NUM    6
#define   CORN   10
#define   RATIO  0.75
#define   AVR    80
#define   NL     3
#define   OL     5
#define   CIRCLE  1
#define   PENT    2
#define   TRIAN   3
#define   SQUARE  4
#define   STAR    5
#define   UKNOWN  6
#define   TOUCH   7

struct object {
        char name;
        float xcoord;
        float ycoord;
        float angl;
};

struct REPLY
  {
    struct STD_RPLY STANDARD;
    struct object part;
  };
```

```
/* extern.c */

/*  External definitions  */


        char    **ip =0;                /* image data */
        int     ix[LENG]=0;             /* list of x-coord. of contour */
        int     iy[LENG]=0;             /* list of y-coord. of contour */
        float   xg = 0.0;               /* x-coord. center of gravity */
        float   yg = 0.0;               /* y-coord. center of gravity */
        float   pi = 3.14159;
        float   adp[CORN]=0;
        int     ixp[CORN]=0;
        int     iyp[CORN]=0;
        int     nap[CORN]=0;


/* fimage.c */

/*
                PARALLEL PROCESSING VERSION
                Interrupt driven VG121
                patched 83 - 10 - 13
                modified to run under new HARMONY 83-11-7
                with "task" replacing "process" and EHVT replacing VAL.

*/

#define         VISION  25
#define         TAXI    18


image ()                                /* object identification */
{
        extern int nap[],ixp[],iyp[];
        extern float xg,yg;

        int i,j,n,np,beta;
        int nbd ;
        int jj;
        float per,p2a,ar,gamma,angle;
        long int line_num;
        long int hpos,vpos;
```

```
    long int pix_val=0x3f;
    float orient();
    int  xgf,ygf,arf,perf,p2af;

    static struct UCB *ucb_it = 0, *ucb_ot = 0;

    struct STD_RQST request;
    unsigned requestor;

    unsigned taxiid;

    struct REPLY reply;
    extern char **ip;

    struct STD_RPLY taxi_reply;


    ucb_it = 0;     /* for restart */
    ucb_ot = 0;
                                    /* open user connections */

    while ( ( ucb_it = _Open ( "POACIA0:", READ ) ) == 0 ) ;
    while ( ( ucb_ot = _Open ( "POACIA0:", WRITE ) ) == 0 ) ;

    _Selectinput( ucb_it );
    _Selectoutput( ucb_ot );        /* connections */

    taxiid = _Create ( TAXI );

/* dynamically allocate ip */

    ip = (char **)_Getvec( (ROW+1)*sizeof( char *) );
    ip[0]=(char *)_Getvec( (ROW+1)*(COL+1) );
    for( i=1; i<=ROW; ++i)
            ip[i] = ip[i-1] + (COL+1) *sizeof(char);

                /* wait for message from TAXI */

    request.MSG_SIZE = sizeof( request );
    requestor = _Receive( (char *)&request, taxiid );

            /* outer loop - receive, reply, take picture ... */


top:
    request.MSG_SIZE = sizeof( request );
```

```
    requestor = _Receive( (char *)&request, 0 );

    jj = 1 ;
    line_num = 0;
    vpos = 0;
    hpos = 0;
    syncmode (1);
    window (line_num);
    pixpos (hpos,vpos);

    taxi_reply.MSG_SIZE = sizeof ( taxi_reply );
    taxi_reply.RESULT = 0;
    _Reply ( (char *)&taxi_reply, taxiid );
    wash();

    taxi_reply.MSG_SIZE = sizeof ( taxi_reply );
    taxi_reply.RESULT = 0;
    _Reply ( (char *)&taxi_reply, taxiid );
    acfield();

    pixpos (hpos,vpos);
    rdfield ( &ip[0][0] );
    pixpos (hpos,vpos);
    threshold();                    /* thresholding */

    taxi_reply.MSG_SIZE = sizeof ( taxi_reply );
    taxi_reply.RESULT = 0;
    _Reply ( (char *)&taxi_reply, taxiid );
    wash();

    pixpos (hpos,vpos);
    wrfield ( &ip[0][0] );
    pixpos (hpos,vpos);
    pad();
    nbd = -1;
                        /* null reply - required for startup */

    reply.STANDARD.MSG_SIZE = sizeof( reply );
    reply.STANDARD.RESULT = nbd;
    _Reply( (char *)&reply, requestor );

    nbd = 0;

/* inner loop - find next boundary */
```

```
contour:

        n = 0;

        bdryb (&jj,&n);                 /* boundary following */

        if ( n > LENG ){
                _Printf ( "list overflowed after bdry # %i \r\n",nbd);
                _Flush ();
                goto contour;
        }
        if ( jj >= ROW-1 ) {
                nbd = 0;
                goto aaa;
        }
        nbd += 1;

        if (nbd == 1)
           {
                taxi_reply.MSG_SIZE = sizeof ( taxi_reply );
                taxi_reply.RESULT = 0;
                _Reply ( (char *)&taxi_reply, taxiid );
                wash();
           }

        per = 0.0;
        acgp (&n,&per,&p2a,&ar);
        xgf = xg * 1000;
        ygf = yg * 1000;
        perf = per * 1000;
        p2af = p2a * 1000;
        arf = ar * 1000;

_Printf (" xg = %i",xgf);
_Printf (" yg = %i",ygf);
_Printf (" n = %i\r\n",n);
_Printf (" area= %i",arf);
_Printf ("   per=%i",perf);
_Printf ("   p2a=%i\r\n",p2af);
_Flush ();

        /* check for unknown objects */
        /* the assumption is that unknown ob. are parts of toy */

        if (abs(ar)<1500 || abs(ar)>5000) goto bbb;
```

```
/* check for touching objects */
/* the assumption is that touching ob. are parts from toy */
if (n > 250 ){
        _Putstr ("Touching Objects\r\n");
        _Flush ();
        reply.part.name=TOUCH;
        reply.part.xcoord=xg;
        reply.part.ycoord=yg;
        reply.part.angl=0;
        goto aaa;
}

/* identify circle */
if (abs(p2a) < 1.045){
        if ( n>155 && n<170 ){
                reply.part.name=CIRCLE;
                reply.part.xcoord=xg;
                reply.part.ycoord=yg;
                reply.part.angl=0;
                goto aaa;
        }
        goto bbb;
}

/* find peaks */
np=0;
angle=0;
n -= 1;
np = peako (&n);
peakad (&np);
peaksd (&np);
_Printf ("np=%i\r\n",np);
_Flush ();

/* identify pentagon */
if (abs(p2a)<1.2 && abs(p2a)>1.05){
        if (np == 5){
                beta=72;
                gamma=54;
                angle=orient(beta,gamma);
                reply.part.name=PENT;
                reply.part.xcoord=xg;
                reply.part.ycoord=yg;
                reply.part.angl=angle;
                goto aaa;
```

```
        }
        goto bbb;
}

/* identify square */
if (abs(p2a)>1.18 && abs(p2a)<1.35){
        if ( np == 4 ){
                beta=90;
                gamma=45;
                angle=orient(beta,gamma);
                reply.part.name=SQUARE;
                reply.part.xcoord=xg;
                reply.part.ycoord=yg;
                reply.part.angl=angle;
                goto aaa;
        }
        goto bbb;
}

/* identify triangle */
if (abs(p2a)>1.35 && abs(p2a)<1.7){
        if ( np == 3 ){
                beta=120;
                gamma=30;
                angle=orient(beta,gamma);
                reply.part.name=TRIAN;
                reply.part.xcoord=xg;
                reply.part.ycoord=yg;
                reply.part.angl=angle;
                goto aaa;
        }
        goto bbb;
}

/* identify star */
if (abs(p2a)>1.9 && abs(p2a)<2.27 && np==5 ){
                beta=72;
                gamma=54;
                angle=orient(beta,gamma);
                reply.part.name=STAR;
                reply.part.xcoord=xg;
                reply.part.ycoord=yg;
                reply.part.angl=angle;
                goto aaa;
}
```

```
bbb:
        _Putstr ("Unknown object\r\n");
        _Flush ();
        reply.part.name=UKNOWN;
        reply.part.xcoord=xg;
        reply.part.ycoord=yg;
        reply.part.angl=0;


    aaa:
        request.MSG_SIZE = sizeof( request );
        requestor = _Receive( (char *)&request, 0 );

        for ( i=0; i<=n; ++i )
        {
                hpos = ix[i];           /* write contour to screen */
                vpos = iy[i];
                wrpixel ( hpos, vpos, pix_val );
        }

        hpos = xg;                      /* write centre of gravity */
        vpos = yg;
        wrpixel ( hpos,vpos,pix_val );

        reply.STANDARD.MSG_SIZE = sizeof( reply );
        reply.STANDARD.RESULT = nbd;
        _Reply( (char *)&reply, _Father_id() );

    /* end of inner and outer loops */
        if (nbd == 0) goto top;
        else  goto contour;
}

error()
{
        _Putstr ("error\n");
        _Flush ();
}


pad()
{
        int i;

        for ( i=0 ; i <= COL ; ++i ){
                ip[0][i] = 0;
```

```
                ip[ROW][1] = 0;
        }
        for ( i=0 ; i <= ROW ; ++i ){
                ip[i][0] = 0;
                ip[i][COL] = 0;
        }
}
```

```
                                /* TAXI */

    /* notifier for vg121 interface */


Taxi ()
{
        struct  STD_RQST        request;
        struct  STD_RPLY        reply;

        for (;;)
           {
                request.MSG_SIZE = sizeof ( request );
                reply.MSG_SIZE = sizeof ( reply );
                _Send ( (char *)&request, (char *)&reply, _Father_id() );
                _Await_Interrupt ( 40 );
           }
}
```

APPENDIX E

Listing of Vision Subroutines

/* thresh.c */

```
threshold()
{
        extern char **ip;
        int i,j;

        for (i=0; i<= ROW ;++i)
                for (j=0; j<= COL ;++j){
                        if ( ip[i][j] > THD )
                                ip[i][j] = 0x3f;
                        else
                                ip[i][j] = 0;
                }
}
```

/* bdryb.c */

```
/* Boundary following algorithm; finds starting point on contour */
/* then calls bdrya.                                              */

bdryb (pjj,pn)
 int  *pjj, *pn;
{

        extern char **ip;
        int  i,j,jp4;
        int  x0,y0;
        int  is=1;

    loop:
        *pn = 0;
        for (i= *pjj; i<=ROW-1; ++i)
                for (j=1; j<=COL-1; ++j){
                        if ( ip[i][j] == 0x3f ){
                           jp4 = ip[i-1][j] * ip[i+1][j] * ip[i][j-1]
                                * ip[i][j+1];
                           if (jp4 == 0) goto track;
                        }
                }
        return;

    track:
        if ( ip[i][j-1] == 0 ) is=3;
        if ( ip[i][j+1] == 0 ) is=7;
        if ( ip[i-1][j] == 0 ) is=5;
        x0 = j;
        y0 = i;
        *pjj = i;
        bdrya (x0,y0,is,pn);
        if ( *pn <= MAXN ) goto loop;
        if ( *pn > LENG )
                return ;
}
```

```
/* bdrya.c */

/* Boundary following algorithm given starting point on contour */


static int  xp[9] = {0,0,-1,-1,-1,0,1,1,1};
static int  yp[9] = {0,1,1,0,-1,-1,-1,0,1};

bdrya (x0,y0,is,pn)
 int  x0,y0,is;
 int  *pn;
{
        extern char **ip;
        extern int ix[],iy[];
        int j=x0;
        int i=y0;
        int kk=1;
        int isw=0;
        int xn , yn;
        int is8,k,in,iw,jw;

        if ( NC == 4 ) kk=2;
        ip[i][j] = 2;
    loop:
        ix[*pn] = j;
        iy[*pn] = i;
        is += kk;
        is8 =is+8;
        for (k=is; k <= is8 ;k += kk ){
                if ((in=k) > 8) in=in-8;
                iw = i + yp[in];
                jw = j + xp[in];
                if ( ip[iw][jw] != 0 ){
                        ip[iw][jw] = 2;
                        goto cntn;
                }
        }
        return;                         /* isolated point */
    cntn:
        is =in + 4;
        if (isw!=1){
                isw = 1;
                xn = jw;
                yn = iw;
        }
        if ( is > 8 ) is = is - 8;
        *pn += 1;
```

```
        if ( *pn > LENG ) return ;
        i = iw;
        j = jw;
        if ( (j!=x0) || (i!=y0) ) goto loop;
        is += kk;
        is8 = is + 8;
        for (k=is; k<=is8; k+=kk){
                if ( (in=k) > 8 ) in -= 8;
                iw = i + yp[in];
                jw = j + xp[in];
                if ( ip[iw][jw] != 0 ) break;
        }
        is -= kk;
        if ( jw != xn || iw !=yn ) goto loop;
        ix [*pn] = j;
        iy [*pn] = i;
        return ;
}
```

/* elextr.c */

```
/* Eliminates extra points from list of boundary; extra points */
/* are those which form a corner between 2 diagonal pts in 2x2 */
/* neighborhood.                                               */


elextr (pn)
int *pn;
{
        extern int ix[],iy[];
        int idx0 = 0;
        int idx = 0;
        int i,inxt;

        for (i=1; i<=*pn; ++i){
                inxt = i+1;
                if ( inxt > *pn ) inxt = 0;
                if ( ix[i]==ix[idx0] && iy[i]==iy[inxt] ) continue;
                if ( ix[i]==ix[inxt] && iy[i]==iy[idx0] ) continue;
                idx0 = idx+1;
                idx += 1;
                ix[idx] = ix[i];
                iy[idx] = iy[i];
        }
        *pn = idx;
}
```

```
/* extr.c */

/* eliminates extra points from list of boundary; extra points */
/* are those which form a path normal to contour               */


extr (pn)
int  *pn;
{
        extern int ix[],iy[];
        int idx0 = 0;
        int  i,inxt;

        for (i=1; i<=*pn; ++i){
                intx = i+1;
                if ( inxt > *pn ) inxt=0;
                if (ix[i]==ix[idx0] && iy[idx0]==iy[inxt]) continue;
                if (ix[inxt]==ix[idx0] && iy[idx0]==iy[i]) continue;
                if (ix[i]==ix[inxt] && iy[inxt]==iy[idx0]) continue;
                if (ix[idx0]==ix[inxt] && iy[inxt]==iy[i]) continue;
                idx0 += 1;
                ix[idx0] = ix[i];
                iy[idx0] = iy[i];
        }
        *pn = idx0;
}
```

/* fold.c */

```
/* eliminates folding points on boundary except of the very first */
/* point                                                          */


fold (pn)
int  *pn;
{
        extern int ix[],iy[];
        int  valid=0;
        int  scan=1;
        int  i;

        for ( i=0; i <= *pn; ++i ){
                if ((ix[scan+1]==ix[valid]) && (iy[scan+1]==iy[valid])){
                        scan += 1;
                        valid -= 1;
                }
                else {
                        valid += 1;
                        ix[valid] = ix[scan];
                        iy[valid] = iy[scan];
                        scan += 1;
                }
        }
        *pn = valid-1;
}
```

/* acgp.c */

```
/* computes area, center of gravity, perimeter and p2a factor from */
/* contour coordinates.                                            */


acgp (pn,pper,pp2a,par)
int *pn;
float *pper,*pp2a,*par;

{
        extern int ix[],iy[];
        extern float xg,yg,pi;
        int k,i;
        int l=5;
        int i2a=0;
        double a,sqrt();

        if ((ix[0] != ix[*pn]) || (iy[0] != iy[*pn])){
                *pn += 1;
                ix[*pn] = ix[0];
                iy[*pn] = iy[0];
        }
        for ( k=0; k<= *pn-1; ++k ){
                i2a += ix[k]*iy[k+1]-iy[k]*ix[k+1];
                 xg += ix[k]*ix[k]*(iy[k+1]-iy[k]);
                 yg += iy[k]*iy[k]*(ix[k+1]-ix[k]);
        }
        xg /= i2a;
        yg /= -i2a;
        *par = i2a/2.;

        for ( i=1; i<= *pn; i+=1 ){
                a = ((double) ((ix[i]-ix[i-1])*(ix[i]-ix[i-1])
                              +(iy[i]-iy[i-1])*(iy[i]-iy[i-1])));
                *pper += sqrt (a);
        }
        i -=1;
        if ( i != *pn )
                *pper += sqrt((double) ((ix[*pn]-ix[i])*(ix[*pn]-ix[i])
                              +(iy[*pn]-iy[i])*(iy[*pn]-iy[i])));
        *pp2a = ((*pper)*(*pper))/(4.*pi*(*par));
}
```

/* peako.c */

```
/* Program computes number of corners and their coordinates   */
/* 1. computes average point of '1' consecutive coord. points */
/*    on contour with '1/2' overlap between the segments of   */
/*    '1' points,                                             */
/* 2. computes distance between average point and center of    */
/*    gravity,                                                */
/* 3. searches the list of distances for peaks.               */


peako (pn)
int *pn;
{
        extern int ix[],iy[],ixp[],iyp[],nap[];
        extern float xg,yg,adp[];
        int  iax[AVR];
        int  iay[AVR];
        int  na[AVR];
        int  j=0, np=0, k=0;
        int  l,ln,ll,m,nm,i,n,kk,ii;
        float  ad[AVR];
        float  sx,sy,sxl,syl,pd;
        double sqrt();

        l = OL/2;
        ll = (OL+1)/2;
        n = *pn + l;
        m = n/ll;
        m *= ll;
        nm= n-m;
        ln = l-nm;
        if (ln != 0){
                for ( i=0; i<=ln-l ; ++i ){
                        ix[n+i]=ix[i];
                        iy[n+i]=iy[i];
                }
        }
        for ( i=0; i<=m-l ; i+=ll ){
                sx=0;
                sy=0;
                j += l;
                for ( kk=i; kk <= i+OL-l ; ++kk){
                        sx += ix[kk];
                        sy += iy[kk];
```

```
                }
                sxl = sx/OL;
                syl = sy/OL;
                ad[j]=sqrt((double) ((xg-sxl)*(xg-sxl)+(yg-syl)*(yg-syl)));
                ii = i+1;
                iax[j] = ix[ii];
                iay[j] = iy[ii];
                na[j] = ii;
        }
        if ( ad[1]>ad[j] && ad[1]>ad[2] ){
                pd = ad[1];
                k = 2;
                goto three;
        }

    ten:                        /* find peaks */
        k += 1;
        if ( k > j ) return (np);
        if ( k == j ) ad [k+1] = ad [1];
        if ( ad [k+1] <= ad [k] ) goto ten;
        pd = 0;
    two:
        if ( pd > ad [k] ) goto three;
        pd = ad [k];
        k += 1;
        if ( k == j+1 ){
                ad [k] = ad [1];
                goto two;
        }
        if ( k > j ) return (np);
        goto two;
    three:
        np += 1;
        adp[np] = pd;
        ixp[np] = iax[k-1];
        iyp[np] = iay[k-1];
        nap[np] = na[k-1];
        goto ten;
}
```

```
/* peakn.c */

/* Program to find number of corners and their coordinates. */
/* Same as peako.c, but no overlap between the consecutive  */
/* segments.                                                */


peakn (pn)
int *pn;
{
        extern int ix[],iy[],ixp[],iyp[],nap[];
        extern float adp[];
        extern float xg,yg;
        int iax[AVR],iay[AVR],na[AVR];
        int j=0,k=0,np=0;
        int m,nm,n,i,kk,ii;
        float ad[AVR];
        float sx,sy,sxl,syl,pd;
        double sqrt();

        n = *pn + 1;
        m = n/NL;
        m *= NL;
        nm = n-m;
        for (i=0; i<=m-1 ; i+= NL){
                sx=0;
                sy=0;
                j += 1;
                for (kk=i; kk <= i+NL-1 ;++kk){
                        sx += ix[kk];
                        sy += iy[kk];
                }
                sxl = sx/NL;
                syl = sy/NL;
                ad[j]=sqrt((double) ((xg-sxl)*(xg-sxl)+(yg-syl)*(yg-syl)));
                ii = NL/2;
                ii += i;
                iax[j]=ix[ii];
                iay[j]=iy[ii];
                na[j]=ii;
        }
        if (nm != 0) {
                sx=0.0;
                sy=0.0;
                j += 1;
                for (kk=m; kk <= *pn ;++kk){
```

```
                        sx += ix[kk];
                        sy += iy[kk];
                }
                sxl = sx/nm;
                syl = sy/nm;
                ad[j]=sqrt((double) ((xg-sxl)*(xg-sxl)+(yg-syl)*(yg-syl)));
                ii = nm/2;
                ii += m;
                iax[j]=ix[ii];
                iay[j]=iy[ii];
                na[j]=ii;
        }
        if ( ad[1]>ad[j] && ad[1]>ad[2]){
                pd = ad[1];
                k = 2;
                goto three;
        }

ten:                            /* find peaks */
        k += 1;
        if ( k > j ) return (np);
        if ( k == j ) ad [k+1] = ad [1];
        if ( ad [k+1] <= ad [k] ) goto ten;
        pd = 0;
two:
        if ( pd > ad [k] ) goto three;
        pd = ad [k];
        k += 1;
        if ( k == j+1 ){
                ad [k] = ad [1];
                goto two;
        }
        if ( k > j ) return (np);
        goto two;
three:
        np += 1;
        adp[np] = pd;
        ixp[np] = iax[k-1];
        iyp[np] = iay[k-1];
        nap[np] = na[k-1];
        goto ten;
}
```

```
/* peakad.c */

/* eliminates false corners from list of peaks _ false corners */
/* are peaks too close to the center of gravity.               */


peakad (pnp)
int *pnp;
{
        extern int ixp[],iyp[],nap[];
        extern float adp[];
        int i,j;
        float adp[CORN];
        float big=0;

        for (i=1; i<= *pnp ;++i ){
                if ( adp[i] > big ) big = adp[i];
        }
        for (i=1; i<=*pnp ;++i ){
                adp[i] = adp[i]/big;
                if (adp[i] > RATIO) continue;
                ixp[i] = 0;
                iyp[i] = 0;
        }
    check:
        for (i=1; i<= *pnp ;++i ){
                if (ixp[i]==0 && iyp[i]==0) goto peak;
        }
        return;
    peak:
        *pnp -= 1;
        for (j=i; j<= *pnp ;++j){
                ixp[j]=ixp[j+1];
                iyp[j]=iyp[j+1];
                adp[j]=adp[j+1];
                nap[j]=nap[j+1];
        }
        goto check;
}
```

/* peaksd.c */

```
/* Eliminates false corners from list of peaks _ a false corner */
/* is one of two peaks too close to each other.                 */
/* This program needs to be rewritten more elegantly.           */


peaksd (pnp)
int *pnp;
{
        extern int ixp[],iyp[],nap[];
        extern float adp[];
        double sqrt();
        float sdp[CORN];
        float big=0;
        int m[CORN];
        int j=0;
        int k,i,nk,l;

        nk= *pnp;
        ixp[nk+1] = ixp[1];
        iyp[nk+1] = iyp[1];
        for (i=1; i<=nk ;++i ){
                sdp[i]=sqrt((double) ((ixp[i+1]-ixp[i])*(ixp[i+1]-ixp[i])
                        +(iyp[i+1]-iyp[i])*(iyp[i+1]-iyp[i])));
                if (sdp[i] > big) big =sdp[i];
        }
        for (i=1; i<= nk ;++i){
                sdp[i] /= big;
                if (sdp[i] <= 0.5){
                        j += 1;
                        m[j] = i;
                        continue;
                }
                if (sdp[i] > 0.85) sdp[i] = 1.0;
        }
        sdp[nk+1] = sdp[1];
        for (i=1; i<=j ; ++i){
                k=m[i];
                l=k-1;
                if (sdp[k+1] == 1.0) goto one;
                if (l == 0) l = nk;
                if (sdp[l] != 1.0) goto one;
                ixp[k+1]=0;
                iyp[k+1]=0;
                if ((sdp[k+1] += sdp[k]) > 0.9){
```

```
                    sdp[k+1] = 1.0;
                    continue;
                }
            one:
                if(ixp[k]==0 && iyp[k]==0) continue;
                ixp[k]=0;
                iyp[k]=0;
                if ((sdp[k] += sdp[1]) > 0.9) sdp[k]=1.0;
        }
    two:
        for (i=1 ; i<=nk ;++i){
                if (ixp[i]==0 && iyp[i]==0) goto peak;
        }
        return;
    peak:                                   /* actual peaks */
        nk -= 1;
        *pnp = nk;
        for (k=i; k<=nk ;++k){
                ixp[k]=ixp[k+1];
                iyp[k]=iyp[k+1];
                adp[k]=adp[k+1];
                nap[k]=nap[k+1];
        }
        goto two;
}
```

/* angor.c */

```
/* Computes the angle 'alpha' between a point on the circumference */
/* of an object and the horizontal, in the clockwise direction.    */


float angor(jx,jy)
int jx,jy;
{
        extern float xg,yg,pi;
        float yn,xd,alpha;
        double arctan();

        yn = yg - jy ;
        xd = xg -jx ;
        alpha = (180/pi)*(arctan((double) (abs(yn)/abs(xd))));
        if ( yn>0 && xd>0 ) return (alpha);
        if ( yn>0 && xd<0 ) return (180.-alpha);
        if (xd<0) return (180.+alpha);
        return (360.-alpha);
}
```

/* orient.c */

```
/* Computes correction angle for part orientation, i.e. the          */
/* difference between a reference angle 'gamma' and the orientation */
/* angle 'alpha'.                                                    */


float orient (beta,gamma)
float gamma;
int beta;
{
        extern int ixp[],iyp[];
        int alp;
        float angle,alpha;
        float angor();

        adjust(2);
        alpha = angor (ixp[2],iyp[2]);
        alp = alpha;
        angle = gamma - (alpha-alp)-(alp%beta);
        return (angle);
}

adjust (k)              /* fine tuning for peak location */
int k;
{
        extern int nap[],ix[],iy[],ixp[],iyp[];
        extern float xg,yg;
        int a,i,j,m=0;
        float pd=0,d[NL];

        a=NL/2;
        i=nap[k];
        for ( j=i-a ; j <= i+a ; ++j){
                d[m] = (xg-ix[j])*(xg-ix[j])+(yg-iy[j])*(yg-iy[j]);
                if (pd <= d[m]){
                        pd = d[m];
                        m += 1;
                }
                else  break;
        }
        ixp[k] = ix[j-1];
        iyp[k] = iy[j-1];
        nap[k] = j-1;
}
```

APPENDIX F

Listing of Robot Application Program

/* fappl.c */

```
/*
            open house demo modified for multiprocessing
            (with EHVT_server)
            and parallel processing on Harmony.
            :
            :   location variables are b   d up on puma's floppy disk
      N.B.  :   #C1182.DG file name: FSH     .
            :
            tool transform OPHOUSE must be invoked.



            1-------socket: circle
            2-------  "     pentagon
            3-------  "     triangle
            4-------  "     square
            5-------  "     star
            6-------reject box
            :
            10------pause
            11------newframe
            12------newseq
            13------boxx
            :
            20------vis_pick
            21------pick_ref
            22------rob_pick
            23------vision
*/
    #include "[harmony.src]templates.h"

    #define VISION          25
    #define LOAFER          50

    #define MAX_FRAM 5
    #define JOINT6 0.0
    #define X_R_PIC 150.0
```

```
        #define Y_R_PIC 120.0

        struct UCB *ucb_it = 0, *ucb_ot = 0;
        extern struct UCB *ucb_iv, *ucb_ov;

main()
{

        struct STD_RQST request;
        struct REPLY    reply;

        unsigned        visionid;

        char cd, c, message[180];

        int frame,nbd,socket,sped=30,x,i,ptrmsg;
        int vac=1,novac= -1,incr=2,stop= -2;
        int pause =10,newframe=11,newseq=12,boxx=13;
        int vis_pick =20,pick_ref =21,rob_pick=22,vision=23;
        int small = 1, longer = 12;
        int joint6 = 6;

        float ref_ang,angle, xscale = .6778, yscale = .684;
        float dx =0,dy =0,dz = -150;
        float xshift,yshift,zshift=0, dist = 150;
        float xpick=100,ypick=100;
        float xc,yc,zc,oc,ac,tc,jt1,jt2,jt3,jt4,jt5,jt6;

        ucb_it = 0;
        ucb_ot = 0;             /* for restart */
        ucb_iv = 0;
        ucb_ov = 0;

        _Create( POACIA0 );
        _Breakpoint ("creation of POACIA0");

        while ( ( ucb_it = _Open ( "POACIA0:", READ ) ) == 0 );
        while ( ( ucb_ot =  Open ( "POACIA0:", WRITE ) ) == 0 ) ;

        _Selectinput( ucb_it );         /* user connections */
        _Selectoutput( ucb_ot );

        _Create ( EHVTP1A1 );
                                        /* open VAL connections */

        while ( ( ucb_iv = _Open ( "EHVT:", READ ) ) == 0 ) ;
```

```
        while ( ( ucb_ov = _Open ( "EHVT:", WRITE ) ) == 0 ) ;

        visionid = _Create( VISION );

        _Putstr ("appl is here\n");
        _Flush ();

/*      _Create ( LOAFER );
*/

   start:                                   /* restart sequence */

     c = 0;
     while ( c != 's')
        {

        _Destroy ( visionid );              /*  remove the remnants  */
        visionid = _Create( VISION );       /*  get a fresh copy  */

        _Put (07); /* beep */
        _Putstr ("\n\n\nSEQUENCE RESTART\n\n");
        _Putstr ("enter 's' to start or\n");
        _Putstr ("'a' to advance the tray \n>");
        _Flush ();

                c = _Get();
                cd = _Get();    /* empty the buffer of a CR */
                _Putstr ("\n");
                _Flush ();

        /*      synchronize link to puma        */

        x = delay (small, message);
_Printf ("delay reply = %i\n", x);
_Flush ();
        if (x != 62) goto start;

        /*      move arm to newseq      */

            x = move (newseq, message);
            if (x != 62)
               {
                   _Putstr ("\nval command failure\n");
                   _Flush ();
                   goto start;
               }
```

```
    if (c == 's') break;        /* run it ! */
    if (c == 'a')
        {
           x = signal (incr, message);
           if (x != 62) goto start;

           x = signal (stop, message);
           if (x != 62) goto start;

           x = delay (longer, message);
           if (x != 62) goto start;
        }
   }


/*  two loops--outer for MAX_FRAM  sectors;inner for 5 blocks*/

 for(frame=1;frame<=MAX_FRAM;++frame)
{
                                /*  tell 'image' to take a picture  */

        request.MSG_SIZE = sizeof( request );
        reply.STANDARD.MSG_SIZE = sizeof( reply );
        _Send( (char *)&request, (char *)&reply, visionid );

        nbd = reply.STANDARD.RESULT;

                /* nbd = -1 after 1st reply only */

        while ( nbd != 0 )
        {
                /* request next part  */

        request.MSG_SIZE = sizeof( request );
        reply.STANDARD.MSG_SIZE = sizeof( reply );
        _Send( (char *)&request, (char *)&reply, visionid );

        nbd = reply.STANDARD.RESULT;

        if ( nbd == 0 ) break;          /*  last/no objects  */

                x = set(vis_pick,pick_ref,message);
                if (x != 62) goto start;

                socket = reply.part.name;
                ref_ang = reply.part.angl;
```

```
xshift = -(reply.part.xcoord -X_R_PIC)* xscale ;
yshift = (reply.part.ycoord -Y_R_PIC)* yscale ;
        x= shift(vis_pick,xshift,yshift,zshift,message);
        if (x != 62) goto start;

        x = setcl (rob_pick, vision, vis_pick, message);
        if (x != 62) goto start;

        x = appro(rob_pick,dist,message);
        if (x != 62) goto start;

        x = where(&xc,&yc,&zc,&oc,&ac,&tc,&jt1,&jt2,&jt3,&jt4,
                &jt5,&jt6);

        if (x != 62) goto start;

        angle = -( jt6 - JOINT6 + ref_ang );

        x = drive(joint6,angle,sped,message);
        if (x != 62) goto start;

        x = draw(dx,dy,dz,message);
        if (x != 62) goto start;

        x = signal(vac,message);
        if (x != 62) goto start;

        x = delay(small,message);
        if (x != 62) goto start;

        x = departs(dist,message);
        if (x != 62) goto start;

        x = appro(socket,dist,message);
        if (x != 62) goto start;

        x = move(socket,message);
/*      ==>             if(x == 45)      panic button
        {
                x = departs(dist,message);
The panic button response       if (x != 62) goto start;
can't be used with
the current hardware    x = move(boxx,message);
configuration.                  if (x != 62) goto start;
```

```
                        x = signal(novac,message);
                        if (x != 62) goto start;

                        x = delay (small, message);
                        if (x != 62) goto start;

                        goto last;
                    }
*/
                        if (x != 62) goto start;

                    x = signal(novac,message);
                    if (x != 62) goto start;

                    x = delay(small,message);
                    if (x != 62) goto start;

                    x = departs(dist,message);
                    if (x != 62) goto start;

        }

            last:
                    x = move(pause,message);
                    if (x != 62) goto start;


/*      sector is now empty     */

            x = move(newframe,message);
               if (x != 62) goto start;

            x = signal(incr,message);
               if (x != 62) goto start;

            x = signal(stop,message);
            if (x != 62) goto start;

            x = delay(longer,message);
            if (x != 62) goto start;


        }
```

```
        x = move(newseq,message);

        goto start;

}

Loaf ()
{
        for (;;)
           {
                _Breakpoint ("LOAFER");
           }
}
```